

(12)特許協力条約に基づいて公開された国際出願

(19) 世界知的所有権機関  
国際事務局(43) 国際公開日  
2004 年 5 月 6 日 (06.05.2004)

PCT

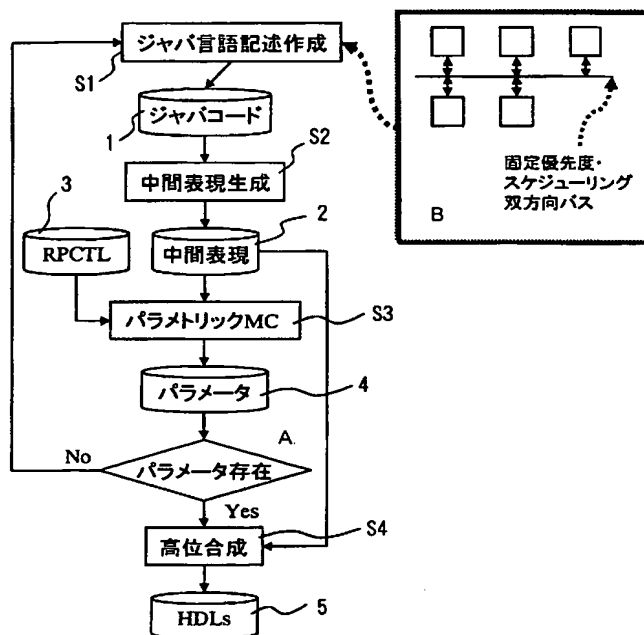
(10) 国際公開番号  
WO 2004/038620 A1

- (51) 国際特許分類<sup>7</sup>: G06F 17/50  
(21) 国際出願番号: PCT/JP2003/012840  
(22) 国際出願日: 2003 年 10 月 7 日 (07.10.2003)  
(25) 国際出願の言語: 日本語  
(26) 国際公開の言語: 日本語  
(30) 優先権データ:  
特願 2002-313201  
2002 年 10 月 28 日 (28.10.2002) JP  
(71) 出願人 (米国を除く全ての指定国について): 株式会社ルネサステクノロジ (RENESAS TECHNOLOGY CORP.) [JP/JP]; 〒100-6334 東京都千代田区丸の内二丁目 4 番 1 号 Tokyo (JP).  
(72) 発明者; および  
(75) 発明者/出願人 (米国についてのみ): 谷本 匡亮 (TANIMOTO, Tadaaki) [JP/JP]; 〒100-6334 東京都千代田区丸の内二丁目 4 番 1 号 株式会社ルネサステクノロジ内 Tokyo (JP). 鎌田 丈良夫 (KAMADA, Masurao) [JP/JP]; 〒100-6334 東京都千代田区丸の内二丁目 4 番 1 号 株式会社ルネサステクノロジ内 Tokyo (JP).  
(74) 代理人: 玉村 静世 (TAMAMURA, Shizuyo); 〒101-0052 東京都千代田区神田小川町 2 丁目 10 番地 新山城ビル 4 2 号 Tokyo (JP).

[続葉有]

(54) Title: SYSTEM DEVELOPMENT METHOD AND DATA PROCESSING SYSTEM

(54) 発明の名称: システム開発方法及びデータ処理システム



S1...JAVA LANGUAGE DESCRIPTION GENERATION  
1...JAVA CODE  
S2...INTERMEDIATE EXPRESSION GENERATION  
2...INTERMEDIATE EXPRESSION  
S3...PARAMETRIC MC  
4...PARAMETER  
A...PARAMETER EXISTS?  
S4...HIGH SYNTHESIS  
B...FIXED PRIORITY/SCHEDULING BIDIRECTIONAL BUS

(57) Abstract: A program description (1) defining a plurality of devices by using a program language capable of describing a parallel operation is input. The program description which has been input is converted into an intermediate expression (S2). A parameter satisfying a real time restriction is generated for the intermediate expression (S3). According to the parameter generated, a circuit description by the hardware description language is synthesized (S4). The intermediate expression is a concurrent control flow flag, a time automaton having a concurrent parameter, or the like. In the aforementioned parameter generation, parametric model checking is performed. The program description is performed by defining a device by using the run method and defining a device clock synchronization by using the barrier synchronization. Thus, it is possible to design a bus system satisfying the real time restriction.

(57) 要約: 並列動作の記述が可能なプログラム言語を用いて複数のデバイスを定義したプログラム記述(1)を入力し、入力したプログラム記述を中間表現に変換し(S2)、中間表現に対し、実時間制約を満足するパラメータを生成し(S3)、生成したパラメータに基づいてハードウェア記述言語による回路記述を合成する(S4)。中間表現は、コンカレントなコントロールフローフラグ、コンカレントなパラメータ付き時間オートマトン等である。前記パラメータ生成に、パラメトリック・モデルチェックを行う。プログラム記述は、runメソッドを用いてデバ

イスの定義を行い、バリア同期を用いてデバイスのクロック同期を定義する。これにより、実時間制約を満たすバス・システムを設計することができる。



(81) 指定国 (国内): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI 特許 (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

添付公開書類:

— 国際調査報告書

(84) 指定国 (広域): ARIPO 特許 (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), ユーラシア特許 (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), ヨーロッパ特許

2 文字コード及び他の略語については、定期発行される各 PCT ガゼットの巻頭に掲載されている「コードと略語のガイダンスノート」を参照。

## 明 細 書

## システム開発方法及びデータ処理システム

## 5 技術分野

本発明は、並列動作を記述できる言語からディジタル回路を開発する方法、更には並列動作を記述できる言語からディジタル回路のハードウェア合成を行うデータ処理システムに関する。

## 10 背景技術

近年、モバイル・コンピューティング環境を実現する上で、システム L S I の果たす役割はその重要性を増している。また、モバイル・コンピューティングでは実時間制約を如何に満たすかが、しばしば問題となる。更に、システム L S I を実装する上で要求性能を満たすよう設計する場合、バス・システムの設計が重要となる。然るに、バス・システムを実時間制約を満たすよう効率良く設計する為の設計手法がシステム・シミュレーションによる方法以外提案されていないのが現状である。システムシミュレーションについて記載された文献の例として下記の特許文献がある。

- 20 特許文献 1 : 特開 2 0 0 2 - 2 7 9 3 3 3 号公報  
特許文献 2 : 特開 2 0 0 0 - 0 3 5 8 9 8 号公報  
特許文献 3 : 特開平 0 7 - 0 8 4 8 3 2 号公報

## 発明の開示

- 25 本発明の目的は、J a v a (登録商標) 等の並列動作を記述可能なプログラム言語を用いてバス・システム等のハードウェア設計の工数を低

減することにある。

本発明の目的は、並列動作を記述可能なプログラム言語とパラメトリック・モデルチェックを用いた、実時間制約を満たすバス・システムの新規設計手法を提供することにある。

- 5      本発明の別の目的は、モデルチェック技術とハードウェア合成技術の融合による新たな設計手法を提供することにある。

- 本発明のさらに別の目的は、実時間制約を有するバス・システムに対する並列動作記述可能なプログラム言語によるモデル化とパラメトリック・モデルチェックによる検証、更にはハードウェア合成を実現
- 10      することにある。

本発明の前記並びにその他の目的と新規な特徴は本明細書の記述及び添付図面から明らかになるであろう。

本願において開示される発明のうち代表的なものの概要を簡単に説明すれば下記の通りである。

- 15      すなわち、並列動作の記述が可能なプログラム言語を用いて複数のデバイスを定義したプログラム記述を入力し、入力したプログラム記述を中間表現に変換し、この中間表現に対し、実時間制約を満足するパラメータを生成し、生成したパラメータに基づいてハードウェア記述言語による回路記述を合成する。

- 20      上記より、Java（登録商標）等の並列動作を記述可能なプログラム言語によりバスシステム等に対するモデル化を行って、実時間制約を満たすシステムの設計を行うことができる。これにより、ハードウェア設計の工数低減が可能になる。

- 本発明の一つの形態として、前記中間表現は、コンカレントなコントロールフローグラフ、コンカレントなパラメータ付き時間オートマトン、又はパラメータ付き時間オートマトンである。
- 25

本発明の一つの形態として、前記パラメータ生成に、パラメトリック・モデルチェックを行う。モデルチェック技術とハードウェア合成技術の融合による新たな設計手法を提供することができる。

5 本発明の一つの形態として、前記実時間制約はR P C T Lで与えられる。

本発明の一つの形態として、前記プログラム記述はr u nメソッドを用いてデバイスの定義を行い、バリア同期を用いてデバイスのクロック同期を定義する。

## 10 図面の簡単な説明

第1図は本発明に係る設計方法の全体を例示するフローチャートである。

第2図は単一バス・システムのジャバ言語によるモデル化のデザインパターンを例示する説明図である。

15 第3図はモデル化におけるクロック同期メカニズムとしてバリア同期によるクロック同期メソッドを用いた例を示す説明図である。

第4図はレジスタへの値書込みを実現するモデル化の記述を例示する説明図である。

第5図はバス権獲得を管理するメソッドを例示する説明図である。

20 第6図は第5図の各メソッドの呼び出し関係を表すコールグラフである。

第7図はバス権獲得メカニズムのジャバ言語記述を例示する説明図である。

第8図は第7図の続きを示すジャバ言語記述の説明図である。

25 第9図はバス権解放を管理するメソッドの説明図である。

第10図は第9図の各メソッドの呼び出し関係を表すコールグラフ

である。

第 1 1 図はバス権解放メカニズムのジャバ言語記述を例示する説明図である。

第 1 2 図は `sync_read` メソッドのジャバ言語コードの説明図である。

5 第 1 3 図は `run()` での `sync_read` メソッド記述例を示す説明図である。

第 1 4 図は `sync_burst_read` のジャバ言語コードを示す説明図である。

第 1 5 図は `endBurstAccess` のジャバ言語コードを示す説明図である。

10 第 1 6 図は `freeBurstBusLock` のジャバ言語コードを示す説明図である。

第 1 7 図は `run()` でのバーストリードの記述例を示す説明図である。

第 1 8 図は `sync_write` のジャバ言語コードを示す説明図である。

第 1 9 図は `run()` での `sync_write` メソッド記述例を示す説明図である。

15 第 2 0 図は `sync_burst_write` のジャバ言語コードを示す説明図である。

第 2 1 図は `run()` でのバーストライトの記述例を示す説明図である。

第 2 2 図はジャバ言語記述による実装例の概略仕様を示す説明図である。

20 第 2 3 図はコマンドインタフェースに関する `run()` method の実装例の一部を示す説明図である。

第 2 4 図は第 2 3 図の実装例の続きを示す説明図である。

第 2 5 図は第 2 4 図の実装例の続きを示す説明図である。

第 2 6 図は第 2 5 図の実装例の続きを示す説明図である。

25 第 2 7 図はユニファイドメモリに関する `run()` method の実装例を示す説明図である。

第 28 図はグラフィック描画ユニットに関する run() method の実装例を示す説明図である。

第 29 図は第 28 図の実装例の続きを示す説明図である。

第 30 図は第 29 図の実装例の続きを示す説明図である。

5 第 31 図は第 30 図の実装例の続きを示す説明図である。

第 32 図は第 31 図の実装例の続きを示す説明図である。

第 33 図は表示ユニットに関する run() method の実装例を示す説明図である。

第 34 図は第 33 図の実装例の続きを示す説明図である。

10 第 35 図は中間表現への変換工程の詳細を全体的に例示する説明図である。

第 36 図は C-CFG の形式を例示する説明図である。

第 37 図は synchronized の扱い（ハード合成用）について示す説明図である。

15 第 38 図は synchronized の扱い（パラメトリック・モデルチェッキング用）について示す説明図である。

第 39 図は Command Interface の場合の C F G を示す説明図である。

第 40 図は Command Interface に関しハード合成用の C F G を例示する説明図である。

20 第 41 図はパラメトリック・モデルチェッキング用の C F G を例示する説明図である。

第 42 図はグラフィック描画ユニット (Graphics Rendering Unit) に関する C F G を例示する説明図である。

第 43 図は第 42 図の C F G の続を示す説明図である。

25 第 44 図はパラメトリック・モデルチェッキング用の C F G を例示する説明図である。

第 4 5 図は第 4 5 図の C F G の続を示す説明図である。

第 4 6 図は前記表示ユニット (Display Unit) に関する C F G を例示する説明図である。

5 第 4 7 図はパラメトリック・モデルチェックング用の C F G を例示する説明図である。

第 4 8 図はコマンドインタフェース (Command Interface) の C F G、グラフィック描画ユニット (Graphics Rendering Unit) の C F G、及び前記表示ユニット (Display Unit) の C F G における夫々の開始ノードの結合状態を例示する説明図である。

10 第 4 9 図は固定優先度スケジューラを例示する説明図である。

第 5 0 図はコマンドインタフェース (Command Interface) の C F G に対する抽象化の様子を順を追って示す最初の説明図である。

第 5 1 図は第 5 0 図の続を示す説明図である。

第 5 2 図は第 5 1 図の続を示す説明図である。

15 第 5 3 図は第 5 2 図の続を示す説明図である。

第 5 4 図は第 5 3 図の続を示す説明図である。

第 5 5 図は第 5 4 図の続を示す説明図である。

第 5 6 図は第 5 5 図の続を示す説明図である。

20 第 5 7 図はグラフィック描画ユニット (Graphics Rendering Unit) の C F G に対する抽象化処理の結果を例示する説明図である。

第 5 8 図は表示ユニット (Display Unit) の C F G に対する抽象化処理の結果を例示する説明図である。

第 5 9 図はコマンドインタフェース (Command Interface) の C F G に対する T N F A への変換の様子を順を追って示す説明図である。

25 第 6 0 図は第 5 9 図の続を示す説明図である。

第 6 1 図は第 6 0 図の続を示す説明図である。



第 6 2 図はグラフィック描画ユニット (Graphics Rendering Unit) の C F G に対する T N F A への変換の様子を順を追って示す説明図である。

第 6 3 図は第 6 2 図の続を示す説明図である。

5 第 6 4 図は第 6 3 図の続を示す説明図である。

第 6 5 図は第 6 4 図の続を示す説明図である。

第 6 6 図は表示ユニット (Display Unit) の C F G に対する T N F A への変換の様子を順を追って例示する説明図である。

第 6 7 図は第 6 6 図の続を示す説明図である。

10 第 6 8 図は第 6 7 図の続を示す説明図である。

第 6 9 図は第 6 1 図、第 6 4 図、第 6 8 図で求めた T N F A の積 T N F A の構成例を示す説明図である。

第 7 0 図はパラメータに上限値を与えた時に、積 T N F A を構成する段階で制約を満たさない遷移枝の削除過程を示す説明図である。

15 第 7 1 図は第 7 0 図の続を示す説明図である。

第 7 2 図は第 7 1 図の続を示す説明図である。

第 7 3 図は第 7 2 図の続を示す説明図である。

第 7 4 図は第 7 3 図の続を示す説明図である。

第 7 5 図は第 7 4 図の続を示す説明図である。

20 第 7 6 図は第 7 5 図の続を示す説明図である。

第 7 7 図は C-TNFA2TNFA で Abstraction of TNFA を適時コールした場合の処理の経過の一部を示す説明図である。

第 7 8 図は第 7 7 図の続を示す説明図である。

第 7 9 図は第 7 8 図の続を示す説明図である。

25 第 8 0 図は得られたパラメータ条件に対して目的関数を各パラメータ値の合計として、これを最小化するという線形計画問題を解いた結果を

例示する説明図である。

第 8 1 図は第 8 0 図の結果に対し、 $kb1=2, kb2=1, kb3=1$  の制約を追加して得た解を例示する説明図である。

5 第 8 2 図は第 8 1 図の結果に対し更に  $kr1$  以外のパラメタ変数を 1 以上という制約を追加して得た解を例示する説明図である。

第 8 3 図は BasicBlock への実行サイクルの割り当てを例示する説明図である。

第 8 4 図は固定優先度スケジューラを例示する説明図である。

10 第 8 5 図は第 8 4 図に示す固定優先度スケジューラの一部を用いて変形した後の C F G を例示する説明図である。

第 8 6 図は変形後の Command Interface の C F G の一部を例示する説明図である。

第 8 7 図は第 8 6 図の続きを示す説明図である。

15 第 8 8 図は孤立ノードへの C F G の割り当てに関する C F G を例示する説明図である。

第 8 9 図は共有レジスタに関し C F G 生成のためのインライン展開の対象になる記述を例示する説明図である。

第 9 0 図は共有レジスタに関する擬似 C 記述を例示する説明図である。

20 第 9 1 図は第 9 0 図の続を示す説明図である。

発明を実施するための最良の形態

本発明の一例として、実時間制約を有するバス・システムの J a v a (登録商標) によるモデル化及びパラメトリック・モデルチェッキング  
25 による検証とハードウェア合成について説明する。本明細書では J a v a (登録商標) を単にジャバ言語とも記す。

## 《設計方法の全体》

第1図には設計方法の全体が示される。設計対象とされるシステムは  
ジャバ言語による記述（ジャバ言語記述）でモデル化される（S1）。  
ジャバ言語記述によるモデル化では、単一バス上のデバイスをジャバ言  
5 語の run()メソッドを用いて記述する。run()メソッドはマルチスレ  
ッドを構成するスレッドにおいて実行させたいプログラムコードがその  
( )内に記述される。クロックはバリア同期を用いて実現される。一般的  
にバリア同期とは複数のモジュールからのデータを受信する場合に同  
時刻に処理されるべき全てのデータを待つための同期手法として把握  
10 することができる。

次いで、S1で生成されたジャバ言語記述（ジャバコード）1を読み  
込み、中間表現に変換する（S2）。ここで、中間表現2は、コンカレ  
ントなコントロール・フロー・グラフ（以下、C-CFG）、コンカレ  
ントなパラメータ付き時間オートマトン（以下、C-TNFA）、パラメ  
15 ータ付き時間オートマトン（以下、TNFA）からなる。CFG（コント  
ロール・フロー・グラフ）は一般に関数内部において制御の流れを示す  
グラムを意味する。TNFA（オートマトン）とは、有限の種類の入力  
を離散的な時刻に受け、過去から現在までに入力された入力の系列を、  
回路で決まった個数以下のクラスに分類して記憶し、それに基づいて有  
20 限の種類 of 出力を出す回路の論理的なモデルとして把握することがで  
きる。

中間表現2として得られたTNFAとRPCTL（Real Time  
Parametric Computation Tree Logic）3で記述された実時間制約を読  
み込み、パラメトリック・モデル・チェックを実行し（S3）、入  
25 力RPCTL等を満たすパラメータ条件4を導出する。

満足するパラメータ条件がなければ方式変更を行いジャバ言語記述

を修正し、満足するパラメータ条件があれば、そのパラメータ条件をサイクル制約として、C-CFGを読み込んで高位合成(S4)によりHDL (hardware Description Language) による回路記述5へ変換する。回路記述は例えばRTL (Register Transfer Level) である。

- 5      上記開発方法はそれを実現するためのプログラムをコンピュータ装置で実行することによって行なわれる。ジャバ言語記述からHDLを得る為の開発支援プログラムはデジタル回路の設計支援プログラムとして位置付けることができる。

- 10      本設計手法により、単一バスシステムの上流工程でのモデル化、検証が可能となり、かつ実時間制約を満たすハードウェアの設計自動化が可能となる。上流工程での検証が可能なのは、設計対象システムはジャバ言語記述でモデル化され、その記述自身が実行可能にされるからである。

《ジャバ言語によるモデル化に対する制約》

- 15      ジャバ言語によるモデル化に際してのジャバ言語記述とバスシステムに対する制約について説明する。単一バス・システムのジャバ言語によるモデル化を目的とするため、ジャバ言語記述に対し、

- 1) ダイナミック・インスタンスエーションの禁止、
- 2) run()メソッドからの start()メソッドコールの禁止、

- 20      の制約を置く。制約1) はLSIのプロセス変更に関するものであり、ここではハードウェアを扱っている事から、許容可能な制約であると考えられる。また、制約2) はバス・プロトコルの検証を目的とする限りに於いては、直接バス動作に関わる部分のみをモデル化すれば良い為、許容可能な制約であると考えられる。

- 25      また、パラメトリック・モデルチェッキング・ツールの制約から、モデルに対して、

- 3) 単一双方向バス・システム、

4) バス権制御は固定優先度

5) バス上の各デバイスは一定周期で処理を終了、の制約を課す。上記制約の緩和は新たな課題として位置付けられる。

《ジャバ言語によるモデル化の為のデザインパターン》

5 第2図には単一バス・システムのジャバ言語によるモデル化の為のデザインパターンが例示される。単一バス・システムのジャバ言語によるモデル化はUML (Unified Modeling Language) で与えられる第2図のデザインパターンに沿って記述される。バス上の各デバイス動作を DeviceImpl Class 内の run() メソッドに実装し、バスを介してアクセスがあるデバイス内レジスタは Register Class 内の attribute (属性) 10 として実装し、バスを介しての同期通信メソッドを Register Class のメソッドとして実装する。また、バスに対応する Bus Class、バスのロック管理を行う BusController Class、及びクロック同期を管理する Clock Controller Class を実装する。バスのロック管理とはバスアービトレーションの制御を意味する。第2図の標記において、棒線を付した三角記号△は継承を意味し、例えば DeviceImpl Class は Device Class 15 の子クラスであり、DeviceImpl Class は Device Class のメソッドを使用可能である。記号→はユース (use) を意味し、例えば Device Class は Clock Controller Class のメソッドを利用する。

20 尚、ジャバ言語コードの再利用性を高めるため、下記方針

- 1) デバイスの追加・削除 (DeviceImpl Class の追加・削除)、
- 2) デバイス動作の変更 (DeviceImpl Class の run() メソッドの変更)、
- 3) 共有変数の追加・削除 (Register Class の attribute の追加・削除)、

25 4) バスプロトコルの追加・変更 (Register Class の同期通信メソッドの追加・変更)、

にて、ジャバ言語コードの変更が可能となるようデザイン・パターンを定義している。

第2図のデザインパターンにおける各クラス (Class) について説明する。

#### 5 (1) Device クラス

デバイスに共通の要素をまとめた抽象クラスである。レジスタやバスは複数ある場合でも処理内容自体が異なることはないため、同じクラスオブジェクトを複数生成ことにより複数存在することを表現できるが、デバイスはデバイスごとにその処理内容が異なる。よって、並列動作を行なうために Thread クラスを継承し、Clock controller Class のインスタンス、Register Class のインスタンス、Bus Class のインスタンスを表すメンバ変数、各デバイスがアクセス可能な共有レジスタの情報を登録する為のメンバ変数、及び各デバイスがアクセス可能な共有レジスタの情報を登録する為のメソッドといったようなデバイスに共通な要素をまとめたクラスとして Device クラスを実装する。

#### (2) DeviceImpl クラス

実際のデバイスにあたるクラスである。デバイスはデバイスごとに処理内容が異なるため、デバイスに必要な要素をまとめた Device クラスを継承し、その処理内容を記述する。つまり、A という処理を行なう DeviceImplA クラス、B という処理を行なう DeviceImplB クラスといったように処理ごとに Device クラスの実装クラスが存在することになる。

#### (3) Register クラス

共有レジスタに対応するクラスである。レジスタの値を表すメンバ変数と、その値をリード/ライトするメソッドから成る。

#### 25 (4) Bus クラス

バスに対応するクラスである。バスが使用中であるか否かの状態を表

すメンバ変数、そのバスに繋がっている共有レジスタをあらわすメンバ変数、及び、状態を変更するメソッドから成る。

#### (5) BusController クラス

バスを介した共有レジスタへのアクセス時のバスロックとバスのロック解放を行なうクラスである。バスのロック・ロック解放はこのクラスに対して依頼する。特に、バスに対して1つデバイスがアクセスを行うとフラグ変数を1に設定するメソッドを含む。このフラグをバスアクセス・フラグとしてメンバ変数として持ち、このメンバ変数により、同一クロック内での複数回のバスロック動作を回避している。

#### 10 (6) ClockController クラス

クロック管理クラスである。バスシステム上の各デバイスにクロック同期動作を行なわせるために、1クロック分の処理終了の通知を集め、全デバイスの処理終了が認められたら、次のクロックの処理を行なってよいという通知を行うと同時に BusController Class のバスアクセス・フラグを0にリセットする。

#### 《モデル化におけるクロック同期メカニズム》

クロック同期メカニズムについて説明する。第3図に示すバリア同期によるクロック同期メソッドを用いて、クロックを実現する。具体的には、1クロック分の処理終了の通知を集め、全デバイスの処理終了が認められたら、次のクロックの処理を行なってよいという通知を行う。

また、バスのロックが残っていない場合には、Bus Class のバスアクセス・フラグを0にリセットする。尚、クロック遷移はパラメトリック・モデル・チェックングへの入力に際して、パラメータ化される為、必ずしも1クロック単位であるとは限らない。

25 このクロック遷移のパラメータ化により、サイクル精度でのモデル化よりも抽象度の高いモデル化を実現する。

レジスタへの値書き込みには1クロックを要する為、それを実現する必要がある。これは、consume\_1\_clock メソッド内で、第4図に例示される Register Class の assignWriteValue メソッドをコールする事で実現している。

#### 5 《モデル化におけるバス権獲得メカニズム》

バス権管理の同期メカニズムとして、先ずバス権獲得メカニズムについて説明する。バス権獲得メカニズムは、第5図に示すメソッドにより、バス権獲得を管理する。バス権の要求は getBusLock メソッドにより行う。第6図には第5図の各メソッドの呼び出し関係を表すコールグラフが示される。第7図及び第8図には上記バス権獲得メカニズムのジャバ言語記述が例示される。

#### 《モデル化におけるバス権解放メカニズム》

次にバス権解放メカニズムについて説明する。バス権解放メカニズムは第9図に示すメソッドにより、バス権解放を管理する。バス権の解放は freeBusLock メソッドにより行う。第10図には第9図の各メソッドの呼び出し関係を表すコールグラフが示される。第11図には上記バス権解放メカニズムのジャバ言語記述が例示される。

#### 《モデル化における排他的同期リード・メソッド》

バスへの排他的同期アクセス方式として、排他的同期リード・メソッドについて説明する。排他的同期リード・メソッドには以下のシングルリードとバーストリードがある。シングルリードは sync\_read メソッドを run() でコールする事で実現される。バーストリードは、sync\_burst\_read、endBurstAccess、consume\_clock メソッドを run() の、synchronized ブロック内で用いる事で実現される。

第12図には sync\_read メソッドのジャバ言語コード、第13図には run() での sync\_read メソッド記述例が示される。run() での記述例に現



れる未定義メソッド、即ち、`do_something_w_or_wo_clock_boundary()`はクロック境界を含んでもよい何らかの処理を意味し、`do_something_wo_clock_boundary()`はクロック境界を含まない何らかの処理を意味する。

5       第14図には `sync_burst_read` のジャバ言語コード、第15図には `endBurstAccess` のジャバ言語コード、第16図には `freeBurstBusLock` のジャバ言語コード、第17図には `run()`でのバーストリードの記述例が示される。バーストリードでは、バスのロックはコールする度に重ねて獲得し、バスの解放を行わないで値を返す。そして、バーストリード  
10   回数のリードが終了すると、重ねたロックを一気に解放するという実装方法となっている。この実装により、毎サイクル、リード値が返ってくるというバースト動作を実現している。

《モデル化における排他的同期ライト・メソッド》

バスへの排他的同期アクセス方式として、排他的同期ライト・メソッドについて説明する。排他的同期ライト・メソッドには以下のシングル  
15   リードとバーストリードがある。シングルライトは `sync_write` メソッドを `run()`でコールする事で実現される。バーストライトは `sync_burst_write`、`endBurstAccess`、`consume_clock` メソッドを `run()`の、`synchronized` ブロック内で用いる事で実現される。

20       第18図には `sync_write` のジャバ言語コードが示され、第19図には `run()`での `sync_write` メソッド記述例が示される。

      第20図には `sync_burst_write` のジャバ言語コードが示され、第21図には `run()`でのバーストライトの記述例が示される。バーストライトでは、バスのロックはコールする度に重ねて獲得し、書き込み処理終了後バスの解放を行わないで処理を終了する。そして、バーストライト  
25   回数のライトが終了すると、重ねたロックを一気に解放するという実装

方法となっている。この実装により、毎サイクル、ライト値が書き込め  
るというバースト動作を実現している。

#### 《ジャバ言語記述による実装例》

ジャバ言語記述による実装例を説明する。第 2 2 図には実装例の概略  
5 仕様が示される。ここでは、共有メモリ方式の 2 次元グラフィックス描  
画・表示システムを一例とする。同図に示されるシステムは、コマンド  
インタフェース (Command Interface)、ユニファイドメモリ (Unified  
Memory)、グラフィック描画ユニット (Graphics Rendering Unit)、  
及び表示ユニット (Display Unit) を有し、それらは双方向バス (Single  
10 bi-derection Bus) に共有する。

(1) Command Interface は、外部からの描画コマンドを受付け、バ  
スを介して受け付けた描画コマンドをメモリへ転送する。

(2) Unified Memory は、描画コマンド、描画ソースデータ、表示  
データを一元的に格納するメモリである。システムの低コスト化・部品  
15 点数削減に効果大。モデルを単純化する為、Java では配列で表現する。  
本デバイスはスレーブデバイスである。

(3) Graphics Rendering Unit は、Unified Memory に描画コマンド  
があるかをポーリングで調べ、存在する場合は、描画コマンド、描画デ  
ータをバスを介してリードしながら描画処理を行い、描画結果を内部バ  
20 ッファに格納し、一気に Unified Memory へバースト転送する。モデル  
を単純化する為、描画コマンド、描画データ転送は配列への連続アクセ  
スで実現する。

(4) Display Unit は、表示データをリードしながら 1 ラインずつ  
表示を行う。モデルを単純化する為、垂直同期はモデル化しない。また、  
25 水平帰線期間はバスアクセスを行わないものとする。

上記コマンドインタフェース (Command Interface)、グラフィック

描画ユニット (Graphics Rendering Unit)、及び表示ユニット (Display Unit) がバスマスタデバイスであり、ユニファイドメモリ (Unified Memory) がバススレーブデバイスである。バスマスタデバイスが同時にバス権獲得を行った場合の、バス権獲得の優先順位は、表示ユニット

5 (Display Unit) > コマンドインタフェース (Command Interface) > グラフィック描画ユニット (Graphics Rendering Unit) とする。

次に、第 22 図の仕様に対する run() method の実装例について説明する。

#### (1) Command Interface

10 先ず、コマンドインタフェースに関する run() method の実装例を説明する。その実装例は第 23 図乃至第 26 図に例示される。外部入力 write\_req 信号 (具体的には CPU からのライト信号) があり且つ wait 出力信号の値決定に用いる内部変数 wait\_flag が false だとコマンド受付を実行 (一定サイクル消費) し、その後に先ず wait\_flag 変数を true

15 とする。次いで、バス権取得を試み、取得できたら Unified Memory のコマンドフラグ 0, 1 を sync\_burst\_read で読み出し、値が 0 (描画コマンド処理済み) であるコマンドフラグに対応するメモリ領域に対して、バス権を取得した状態のままで、sync\_burst\_write を実行し、コマンドフラグ、描画コマンド、描画ソースデータを転送し、wait\_flag 変数

20 を false とする。但し、モデル簡略化の為、例えコマンドフラグが両方とも 0 であっても、コマンドフラグ 0 に対応するメモリ領域への sync\_burst\_write による書き込みのみを実行するものとし、描画ソースデータの転送は省略する。

write\_flag 変数が true で、write\_req 入力信号が true の場合のみ、

25 wait 信号を true とし、それ以外は wait 信号を false とする。特に、コマンド受付実行中は wait\_flag 変数を false とする。尚、wait 信号、

wait\_flag 変数ともに初期値は false とする。

また、Command Interface が受け付けるコマンドは、コマンドフラグ、描画コマンド、描画ソースデータ、テクスチャデータの格納開始・終了アドレスからなっており、テクスチャデータの格納開始・終了アドレスは描画コマンドに含まれているものとする。

## (2) Unified Memory

ユニファイドメモリに関する run() method の実装例を説明する。その実装例は第 27 図に例示される。グラフィックス描画コマンド、描画ソース・グラフィックス・データ、描画後の表示用データを一元的に格納しているメモリ。モデル化を簡略化する為、何も実行しないスレッドとして実現。このメモリの実体は Register Class のインスタンスとして run() メソッド内でアローケートされている。

尚、モデル簡略化の為、メモリ配列を下記

mem\_con\_reg.current\_value[0] : コマンドフラグ 0、

mem\_con\_reg.current\_value[1] : 描画コマンド 0、

mem\_con\_reg.current\_value[2] : コマンドフラグ 1、

mem\_con\_reg.current\_value[3] : 描画コマンド 1、

mem\_con\_reg.current\_value[4] : 描画ソース・データ、及び描画後の表示用データ、の構造としている。

実際には、描画ソース・データと描画後の表示データを格納している配列のなすメモリ空間を 2 分し（一方を Buffer0、他方を Buffer1 とする）、フレーム表示毎にメモリ配列を、

(Buffer0, Buffer1) = (描画ソース・データ, 描画後の表示用データ)

(Buffer0, Buffer1) = (描画後の表示用データ, 描画ソース・データ)

となるように、交互に切り替えているとする。従って、描画ソース・データを描画後の表示用データで上書きする事はないとする。更に、モデ

ル簡略化の為、描画ソース・データの転送は省略する。

また、モデル簡略化の為に、テクスチャ・データは省略しており、描画コマンドも最高で2つしか格納しないものとし、コマンド0, 1の実行順番がどうであっても描画結果には影響しないものとする。

#### 5 (3) Graphics Rendering Unit

グラフィック描画ユニットに関する run() method の実装例を説明する。その実装例は第28図乃至第32図に例示される。Unified Memory のコマンドフラグ0, 1をある一定周期毎に sync\_burst\_read により読み出す(バス権獲得・解放動作)。何れか一方の値が1であった場合、  
10 内部状態変数 render\_start を true とし、バス権獲得を試み、獲得に成功したなら、描画コマンド、描画データを sync\_burst\_read で読み込みながら、描画処理を実行し、描画結果を内部バッファに書き込む(読み込み終了次第、コマンドフラグを0にクリアし、バス権解放する)。但し、モデル簡略化の為、例えばコマンドフラグが両方とも0であっても、  
15 コマンドフラグ0に対応する描画コマンドのみを実行するものとし、描画ソースデータの転送は省略する。再び、バス権獲得を試み、獲得に成功したなら、描画結果を sync\_burst\_write で Unified Memory へ転送し、バス権を解放し、内部状態変数 render\_start を false とする。尚、内部状態変数 render\_start の初期値は false である。

#### 20 (4) Display Unit

表示ユニットに関する run() method の実装例を説明する。その実装例は第33図及び第34図に例示される。Unified Memory から描画後の表示用データを読み込みCRTへ出力する。但し、モデル簡略化の為、水平同期のみを扱うものとする。これは、水平帰線期間内に描画が開始・終了する為のサイクル制約をパラメトリック・モデルチェックで求める事を目的としているからである。  
25

さて、Display Unit は、内部変数 `display_start` を `true` に設定し、Unified Memory から描画後の表示用データを `sync_burst_read` にて読み込む。データ読み込み終了後に内部変数 `display_start` を `false` として、ライン表示を実行し、ライン表示終了後から水平帰線期間に相当する  
 5 適当なサイクルを消費し、再び、ライン表示を同じ手順で行うものとする。また、転送サイクルのみをモデル化すれば良いので、メモリアドレスは簡略的に表現し、常に同一メモリ空間から読み出しているものとする。

#### 《パラメトリック解析》

10 ここでパラメトリック解析の目標について説明する。本来、描画は 1 フレーム表示の表示期間及び、垂直帰線期間の間に 1 フレーム分の実行を終了しておれば良いと考えられるので、水平帰線期間での描画が満たすべき性質を勘案すると、検証内容としては下記条件を満たすサイクル制約の導出で十分である。即ち、「表示用データ取得終了後に描画を開始し、次の表示開始までに描画を終了する事がある。但し、描画開始・  
 15 終了にデータ転送サイクルは含まれ、表示は  $n$  サイクル以内に終了する。」である。上記制約を R P C T L (Real Time Parametric Computation Tree Logic) で記述すると下記

```

EF{<display_end>(AF(<render_begin>true)
20 &&{AF(<render_end>true)AU(<display_begin>true)}}}
&& AG{[display_begin](AF<=n([display_end]true))}

```

の 3 個の制約のアンド条件となる。尚、表示データ取得終了迄に未実行の描画コマンドが Unified Memory 上に常に存在する事が仮定できるものとする。ここで、 $n$  は適当な正整数であり、各変数は下記の通り、  
 25 `display_begin`: 内部変数 `display_start` の 0 から 1 への立ち上がりイベント

display\_end: 内部変数 display\_start の 1 から 0 への立ち下がりイベント

render\_begin: 内部変数 render\_start の 0 から 1 への立ち上がりイベント

5 render\_end: 内部変数 render\_start の 1 から 0 への立ち下がりイベント

を表す。

パラメトリック・モデルチェックングを高速化する目的で、R P C T L を

10  $AG\{[display\_begin](AF\leq n([display\_end]true))\}$  (\*)

と、それ以外の部分に分離し、例えば、

$EF\{<display\_end>(EF(<render\_begin>true)\&\{EF(<render\_end>true)AU(<display\_begin>true)\})\}$

から実行し、パラメータ条件を導出し、導出されたパラメータから (\*)

15 の n の上限を求め、既に求めたパラメータ条件に矛盾しないパラメータ条件が導出できるまで、n の値を減じて繰り返し、(\*) のパラメトリック解析を実施するものとする。

R P C T L やそれを用いたパラメトリック・モデルチェックングの詳細は次の文献、“Akio Nakata and Teruo Higashino: ”Deriving  
20 Parameter Conditions for Periodic Timed Automata Satisfying Real-Time Temporal Logic Formulas”, Proc. of IFIP TC6/WG6.1 Int’l Conf. on Formal Techniques for Networked and Distributed Systems(FORTE2001), Cheju Island, Korea, Kluwer Academic Publishers, pp.151-166, Aug. 2001.”に記載ある。

25 《中間表現への変換》

第 3 5 図には中間表現へ変換工程の詳細が全体的に例示される。バス

システムをモデル化して記述されたジャバ言語記述は C (concurrent) - C F G に変換され (Java2C-CFG)、C - C F G は C - T N F A に変換され (C-CFG2C-TNFA)、C - T N F A は T N F A に変換される (C-TNFA2TNFA)。パラメトリック・モデルチェッキングを経て C - C F G が H D L に変換される (C-CFG2HDL)。尚、上記 Java2C-CFG 等の標記において、数字 “2” は “to” を意味するものと理解されたい。

#### 《Java 2 C-CFG》

ジャバ言語記述から C - C F G への変換アルゴリズムの概要を説明する。各 run() メソッドに対応する C F G を生成する。特に、クロック同期メソッド (consume\_clock) はクロック境界として識別し、呼び出しメソッドは呼び出し関係を表すノードを用いて表現する。各 run() メソッドに対応する C F G が生成されると、fork ノードを設け、そのノードから各 C F G の開始ノードへの fork 枝を付加する。特に、バス上の各デバイスに run() メソッドが対応するものとして Java が記述されている事を前提とし、メモリデバイスに関しては、C F G 生成対象外である事の指定を与えるものとする。

次いで、呼び出しメソッドの C F G を作成し、呼び出しメソッドが synchronized ブロック内に存在する場合は、呼び出しメソッド内の synchronized を C F G から削除する。特に、呼び出しメソッドが Register クラス内の通信メソッドである場合は C F G の作成を行わず、インスタンス関係からどのデバイス内のどのレジスタへのアクセスかと通信メソッドの名前だけを識別し、その情報を C F G に保持する。但し、図ではどのレジスタへのアクセスかの情報は省略している。尚、通信メソッドが内部にクロック同期メソッドを含む場合は、その出力枝にクロック境界をマークする。更に、呼び出しメソッド全体のサイクル制約を導出する以外は、呼び出しメソッドをインライニングする。この例



では、Rendering メソッド、Display メソッドはインライニングしない。

synchronized を予め定めた C F G に展開し、ハード合成用の固定優先度スケジューラを F S M (Finite State Machine) の形式でスケルトンとして予め与えてあるライブラリと実際に C F G を生成した run()  
5 メソッドの個数を用いて、自動生成する。ここで、C F G はハード合成用とパラメトリック・モデルチェック用の 2 種作成する。

尚、固定値伝播や、代入によるローカル変数消去等のコードレベル最適化を C F G 上で実施する。特に、信号の入力・出力・入出力はメンバ変数で表し、端子方向に関しては、プログラム内での代入文で右辺にあるのか左辺にあるのかや条件文での変数の用いられ方を解析して決定  
10 する。入出力信号として識別された場合、データ依存性を考慮して、入力と出力に適当な信号のリネームを行って分離する。

Java 2 C-CFG に関し、以下で、Java 記述例に沿って、各 run() メソッドの C F G 生成結果、C - C F G 生成結果、及び固定優先度・スケジューラの F S M について説明する。  
15

先ず C F G の形式について説明する。第 3 6 図には C - C F G の形式が例示される。各 C F G は分岐枝の開始と終了を表すノードと、ループ枝の開始と終了を表すノードを含み、分岐条件、ループ終了条件を対応する枝に付加した形式とする。特に、クロック境界ノードを枝上に付加する事でクロック境界を表現する。また、メソッドコールに対応するノードを用いる事で、サブ C F G とのリンクを表現する。各 C F G の並列実行を表す fork ノードは C F G への頂点へ付加する事で表現する。  
20

synchronized の扱い (ハード合成用) について説明する。第 3 7 図に例示されるように、synchronized の開始を C F G 上で begin\_sync とラベル付けされたノードとして表し、synchronized の終了を end\_sync  
25 とラベル付けされたノードでし、C F G を生成。その後、その両ノード

を夫々下図に示すように変換する。

synchronized の扱い（パラメトリック・モデルチェック用）について説明する。第 38 図に例示されるように、synchronized の開始を C F G 上で Begin\_sync とラベル付けされたノードとして表し、  
5 synchronized の終了を End\_sync とラベル付けされたノードとし、C F G を生成。その後、その両ノードを夫々第 38 図に示すように変換する。

上記に従うと、前記 Command Interface の場合、C F G は第 39 図に示される。getWriteSignal()メソッドは、シミュレーションを実施する為に行った記述であり、処理系への入力では外部からの入力信号  
10 write\_req であるとして修正されているものとし、入力テストベクタの個数を表す dcom\_index を補正する記述も削除されているものとする。  
また、drawing\_commands も配列として表現されているが、これもシミュレーションを実施する為になされたものであり、内部変数 input\_command に入力変数 drawing\_commands が入力されている記述が  
15 処理系に入力されているものとして扱う。

前記 Command Interface に関し、ハード合成用の C F G は第 40 図に示され、パラメトリック・モデルチェック用の C F G は第 41 図に例示される。

前記グラフィック描画ユニット (Graphics Rendering Unit) に関し、  
20 第 42 図及び第 43 図にはその C F G が例示され、第 44 図及び第 45 図にはそのパラメトリック・モデルチェック用の C F G が例示される。

前記表示ユニット (Display Unit) に関し、第 46 図にはその C F G が例示され、第 47 図にはそのパラメトリック・モデルチェック用の  
25 の C F G が例示される。

第 48 図にはコマンドインタフェース (Command Interface) の C F

G、グラフィック描画ユニット (Graphics Rendering Unit) の C F G、及び前記表示ユニット (Display Unit) の C F Gにおける夫々の開始ノードの結合状態が例示される。

- 5 固定優先度スケジューラ (Fixed Priority Scheduler) について説明する。実際にトップレベルの C F G を生成した run メソッドの数を識別し、夫々のデバイスにバス権取得通知を行うための信号 locked\_i の組みからなるステートを作成する。単一バスシステムであるため、それらステートは、1つの locked\_i のみが1でその他が0であるものを構成すれば良い。また、全ての locked\_i が0であるステートも作成する。
- 10 優先度情報に基づき、バス権要求信号 lock\_i を受けて行う状態遷移枝を設ければバス権管理用の固定優先度スケジューラが構成できる。特に、全ての locked\_i が0であるステートをリセット解除時の開始ステートとする。また、各状態遷移は  $1 \leq t \leq k1$  の時間制約が付加されているものとする。このパラメータは後のパラメトリック・モデルチェックング
- 15 で決定されるパラメータである。尚、例えば、 $k1=2$  とし、各遷移を2クロック遷移とした場合は、出力値は変化が起こるまで、前の値を保持しているものとして解釈するものとする。

第49図には本例に対応する固定優先度スケジューラが例示される。同図における信号の意味は、

- 20 lock\_1 : Command Interface からのバス権要求信号  
lock\_2 : Graphics Rendering Unit からのバス権要求信号  
lock\_3 : Display Unit からのバス権要求信号  
locked\_1 : Command Interface へのバス権取得通知信号  
locked\_2 : Graphics Rendering Unit へのバス権取得通知信号
- 25 locked\_3 : Display Unit へのバス権取得通知信号
- である。尚、バス権取得の優先度が、

Display Unit > Command Interface > Graphics Rendering Unit

であるので、遷移条件は、

lock\_3 : Display Unit への遷移

!lock\_3&&lock\_1 : Command Interface への遷移条件

5      !lock\_3&&!lock\_1&&lock\_2 : Graphics Rendering Unit への遷移条件

otherwise : 全ての locked\_i が 0 である状態への遷移条件

となる。

#### 《Abstraction of each CFG》

夫々の C F G の抽象化処理について説明する。先ず、そのアルゴリズムの概要を説明する。各 BasicBlock に対して、 $0 \leq t \leq k_i$  なる時間遷移条件を付加し、Begin\_sync ノードと End\_sync ノードに対して  $0 \leq t \leq k_l$  を付加し、通信メソッドを表すノードに対して、バースト開始に  $1 \leq t \leq k_{b1}$ 、バースト動作に  $0 \leq t \leq k_{b2}$ 、バースト終了に  $0 \leq t \leq k_{b3}$  を付加する。但し、通信メソッドを表すノードに対しては、バースト開始が 2 サイクル以上、バースト動作時は 1 サイクル以上、バースト終了が 1 サイクル以上を要すると考え、後述のノードのマージによる抽象化を行う際には、サイクル制約の補正を行う。Begin\_sync と End\_sync ノード以外の制約を付加したノードをクロック境界と見做して抽象化を行う。

20      通信メソッドは動作内容を全て抽象化し、連続する場合は、1 つのノードにまとめ、例えばバースト開始 ( $1 \leq t \leq k_{b1}$ )、クロック境界 ( $t=1$ )、(バースト動作 ( $0 \leq t \leq k_{b2}$ ) + クロック境界 ( $t=1$ )) 3 回、バースト終了 ( $0 \leq t \leq k_{b3}$ )、クロック境界 ( $t=1$ ) とすると、

$$6 \leq t \leq (k_{b1}+1-1)+3(k_{b2}+1-1)+(k_{b3}+1-1) = k_{b1}+3k_{b2}+k_{b3} \quad (>=2+3*1+1 =$$

25      6)

なるサイクル制約を持つクロック境界として 1 つにまとめる。また、分

岐ノードは全て非決定分岐であるとする。Basic Block ノードに関しては、入力 RPCTL で用いる変数のみ残し他は抽象化する。分岐ノード下位に存在する Basic Block に対応する部分が異なる分岐で同じ場合は、1つを残して他を削除する。

- 5 特にループノードに関しては、break、continue を含まない固定回数ループの場合はアンローリングを実施し、それ以外の場合は、ループを抜ける条件枝の直前に全体で  $0 \leq t \leq kloop$  のサイクル制約を持つような C F G ノードの集合を設ける事でループを削除する。ここで、kloop の満たすべき条件を別に算出するものとする。更に連続するクロック境界はパーコレーション・ベースの移動を行う事で、1つに纏め、遷移サイクル制約を更新する。
- 10

ここで、各クロック境界のパラメータ化により、クロック境界に与えた遷移サイクル制約の意味は、クロック境界から次のクロック境界への遷移に対するサイクル制約である。

- 15 コマンドインタフェース (Command Interface) の C F G に対する抽象化の様子が順を追って第 50 図乃至第 56 図に例示される。第 57 図にはグラフィック描画ユニット (Graphics Rendering Unit) の C F G に対する抽象化処理の結果が例示される。第 58 図には表示ユニット (Display Unit) の C F G に対する抽象化処理の結果が例示される。

20 《C-CFG2C-TNFA》

C-TNFA から TNFA への変換処理について説明する。TNFA とは時間オートマトンであり、各遷移が、

$s \xrightarrow{a@t[P(t)]} s'$

の形式で表されるものである。ここで、各記号の意味は、

- 25  $s, s'$  : 状態、  
 $a$  : 動作(省略可能)、

@?t: 状態  $s$  を訪れてから動作  $a$  を実行するまでの経過時間を  $t$  に代入、  
 $P(t)$ : 時間制約( $t$  に関する線形不等式の論理結合)、  
である。また、その意味は「 $s$  から  $t$  単位時間経過後に状態  $s'$  に遷移。  
ただし、 $t$  は時間制約  $P(t)$  を満たす場合に限る」である。

5         $C-TNF A$  は、複数の  $TNF A$  が並列動作するモデルであり、動作  
sync は高々 1 つの  $TNF A$  しか実行できないモデルである。特に連続  
する動作 sync に対しては他の  $TNF A$  が割り込めない。尚、各  $TNF A$   
に対して、初期状態から初期状態への遷移に対して、上限を与える為  
の時間制約を課す事が可能である。

10        次に  $C-CFG$  から  $C-TNF A$  への変換処理について説明する。先  
ず、その変換アルゴリズムの概要を説明する。Begin\_sync、End\_sync  
で挟まれた部分を識別し、sync ブロックとする。Sync ブロック内に存  
在しないクロック境界ノードにステート割り当て候補とし、また sync  
ブロック内であっても検証性質に必要な信号が Basic Block として  
15        与えられている場合はその前後のクロック境界ノードにステートを割  
り当て候補とし、最後に Begin\_sync、End\_sync の変換で導入したクロ  
ック境界をステート割り当て候補とする。得られたステート割り当て候  
補を重複が無いようにし、ステート割り当てを行い、各ステートからス  
テートまでを D F S (Depth First Search) でトラバースする事で  $TNF$   
20         $A$  への変換を行う。得られた  $TNF A$  に対して、sync ブロックに対  
応する遷移を検出し、sync 遷移とする。連続する sync 遷移でない遷移  
は 1 つの遷移に纏める事でステート数の削減を行う。

25        コマンドインタフェース (Command Interface) の  $CFG$  に対する  $TNF A$  への変換の様子が順を追って第 59 図乃至第 61 図に例示され  
る。

第 62 図乃至第 65 図にはグラフィック描画ユニット (Graphics

Rendering Unit) の C F G に対する T N F A への変換の様子が順を追って例示される。第 6 5 図において、検証性質は、「表示用データ取得終了後に描画を開始し、次の表示開始までに描画を終了する事がある。但し、描画開始・終了にデータ転送サイクルは含まれ、表示は n サイクル以内に終了する。」であり、これは、描画が開始される事がもしあればという前提にたった検証性質であるから、Graphics Rendering Unit に対応する T N F A において、表示データ取得終了迄に未実行の描画コマンドが Unified Memory 上に存在しない事を表す R2 から R1 への遷移はこの検証には不要となる。従って、これは削除されている。

第 6 6 図乃至第 6 8 図には表示ユニット (Display Unit) の C F G に対する T N F A への変換の様子が順を追って例示される。

#### 《C-TNFA2TNFA》

C - T N F A から T N F A への変換アルゴリズムの概要を説明する。先ず、下記方針 (1) ~ (5) に従って C - T N F A から積 T N F A を構成する。

(1) sync 動作とそれ以外の動作は互いにインタリーブされる。ここで、sync 動作を行っている間にバス権をとる必要が無い動作、即ち sync でない動作を (一般に複数) 並行して実行できることを表現。但し、sync 動作を 1 遷移実行する間の他の TNFA の並行遷移に遷移枝を通る回数に制限を設け、それを満たす各遷移の遷移時間の上限を導出する。特に、回数は 1 から始め、意味のある解が得られるまで、回数を 1 ずつインクリメントするものとする。

(2) 複数の sync 動作が実行可能な場合は静的優先度が最も高いものによって遷移する。ここで、動いていない状態 s に関しては  $\text{age}(s, t)$  (状態 s で時間 t だけが経過した状態) に遷移。ただし、連続する sync 遷移の間には他の動作は割り込めない。

(3)  $\text{age}(s, t)$ 以降の遷移に関しては、時間  $t$  に関して下記補正を行う。即ち、 $s - [P(t_1)] \rightarrow s'$  ならば  $\text{age}(s, t) - [P(t+t_1)] \rightarrow s'[t+t_1/t_1]$ 、とする。ここで、 $s[e/t]$  : 状態  $s$  からの遷移条件に現れる変数  $t$  を式  $e$  で置き換えることを表す。尚、再構成した  $P(t)$  が矛盾した式になっている場合、その遷移は削除し、その遷移へのみ到達する状態全てを削除するものとする。

(4) 各遷移の時間変数  $t$  は下記に示すように、異なる名前に変更する。即ち、 $s - [P(t)] \rightarrow s' - [Q(t)] \rightarrow s''$  は  $s - [P(t_1)] \rightarrow s' - [Q(t_2)] \rightarrow s''$  に変更する。 $\text{age}(s, t)$  に関する補正によって、各遷移条件はそれ以前の遷移の時間変数も含む式になるため、それらを区別するために、この変更が必要となる。

(5) 上限を与えた TNFA がある場合、構成後の TNFA でその初期状態を含む状態間の遷移がその上限を満たさない場合、その途中にある状態で上限を満たす遷移内に存在する状態を削除対象外とし、削除対象外とならなかった状態を削除する。

尚、Abstraction of each CFG のステップで抽象化対象から除外された R P C T L に用いられている変数を伴う遷移が現れるとその次の状態まで構成し、上記構成方針での作成での切りの良いところで、Abstraction of TNFA をコールし、構成中の TNFA の抽象化を行う。

次に C - TNFA から TNFA への変換処理を詳細に説明する。まず、遷移回数の仮定による遷移時間の上限を決定する。即ち、各 TNFA の強連結成分を求め、強連結成分内での遷移枝を 1 回のみ通る各遷移の下限遷移時間の総計が最大となる最長パスと、そのパス上の頂点でその後段の頂点への遷移時間の下限が最小の頂点を求め、最長パスにその後段頂点へのパスを追加したパスの 2 つを求める。次いで、夫々のパスの下限遷移時間の総計を求める。これにより、



Command Interface : 22 23

Graphics Rendering Unit : 28 29

Display Unit : 11 12

5 が求まる。さて、各 T N F A の 2 つ目の下限遷移時間の総計から 1 引いた値、

Command Interface : 22

Graphics Rendering Unit : 28

Display Unit : 11

10 を求める。各 T N F A の各遷移の上限時間を、他の T N F A に対して求めた値の最小値を与える事で決定し、得られた時間遷移の上限値が矛盾したものとなっていないかを、既に上限が与えられている遷移の上限値と比較する事で調べ、もし、矛盾がでたなら 2 回に増加させる。そうでなければ、上限が与えられていない遷移に求めた上限を与える事で線形不等式の論理結合を構成する。特に、回数を増加させる場合、最長パス  
15 の下限遷移時間の総計と回数の積を取った値と、その値に先に求めた最小の遷移時間を加えた値を算出すればよい。

尚、パラメトリック・モデルチェッキングを実施し、パラメータ条件が非負整数解を持たない場合にも回数の増加を行って処理をすすめる事とする。ここで、非負整数解の算出には、線形計画法を用いる。具体的には、フリーソフト LP-SOLV を用いて得られたパラメータ条件を満たす範囲で、パラメータの和が最小となる解を算出する。仮に、得られた解が意味をなさない場合は、パラメータの最小値を付加するなどして対処する。

20 さて、本例の場合、遷移枝を一回のみ遷移可能とした仮定では、各 T N F A の遷移時間の上限として

Command Interface : 11

Graphics Rendering Unit : 11

Display Unit : 22

が求まる。また、矛盾が起こることなく、以下に示す線形不等式の論理結合、

$$\begin{aligned}
 5 \quad & 0 \leq k_1 + k_2 + k_{loop1} \leq 7 \quad \&\& \quad 5 \leq kb_1 + kb_2 + kb_3 + k_l \leq 8 \quad \&\& \\
 & 7 \leq kb_1 + 3kb_2 + kb_3 + k_l \leq 11 \&\& 1 \leq k_l \leq 11 \&\& 1 \leq k_1 + k_2 + k_3 + k_{loop1} + k_l \leq 6 \&\& \\
 & 3 \leq k_4 \leq 11 \&\& 5 \leq kb_1 + kb_2 + kb_3 + k_l \leq 11 \&\& 6 \leq kb_1 + 4kb_2 + kb_3 + 3kr_1 \leq 12 \&\& \\
 & 1 \leq 3kr_2 + k_l \leq 8 \&\& 9 \leq kb_1 + 5kb_2 + kb_3 + k_l \leq 12 \quad \&\& \quad kr_1 = kb_2 - 1 \quad \&\& \\
 & 8 \leq kb_1 + 5kb_2 + kb_3 \leq 24 \&\& 1 \leq k_l + 6kd \leq 19
 \end{aligned}$$

10  が得られる。また、遷移枝を 2 回まで遷移可能とした場合、各 T N F A の遷移時間の上限として

Command Interface : 22

Graphics Rendering Unit : 22

Display Unit : 44

15  が求まり、以下に示す線形不等式の論理結合、

$$\begin{aligned}
 & 0 \leq k_1 + k_2 + k_{loop1} \leq 18 \quad \&\& \quad 5 \leq kb_1 + kb_2 + kb_3 + k_l \leq 19 \quad \&\& \\
 & 7 \leq kb_1 + 3kb_2 + kb_3 + k_l \leq 22 \&\& 1 \leq k_l \leq 22 \&\& 1 \leq k_1 + k_2 + k_3 + k_{loop1} + k_l \leq 17 \\
 & \&\& 3 \leq k_4 \leq 2 \&\& 5 \leq kb_1 + kb_2 + kb_3 + k_l \leq 22 \&\& 6 \leq kb_1 + 4kb_2 + kb_3 + 3kr_1 \leq 23 \\
 & \&\& 1 \leq 3kr_2 + k_l \leq 19 \&\& 9 \leq kb_1 + 5kb_2 + kb_3 + k_l \leq 23 \quad \&\& \quad kr_1 = kb_2 - 1 \quad \&\& \\
 20 \quad & 8 \leq kb_1 + 5kb_2 + kb_3 \leq 46 \&\& 1 \leq k_l + 6kd \leq 41
 \end{aligned}$$

が得られる。

この工程は、Assume Guarantee Reasoning と呼ばれる手法であり、パラメトリック解析での実施は公知ではない。

25  第 6 9 図には第 6 1 図、第 6 4 図、第 6 8 図で求めた T N F A の積 T N F A の構成例が示される。これに基づいて T N F A を求める過程が第 7 0 図乃至第 7 6 図に示される。

## 《Abstraction of TNFA》

TNFAの抽象化処理について説明する。先ず、そのアルゴリズムの概要を説明する。TNFAで抽象化時に残した変数への代入を行っている遷移枝と、その遷移枝の始点ノード及び終点ノードのみを残し、残すべき頂点を始点として、次に残すべき頂点が現れるまで頂点と辺をトラバースしながら遷移時間を再計算する事で、その他の頂点を全て抽象化する。但し、葉に対応する状態は抽象化の対象から除外する。

先の例では抽象化の実施がまだ起こらない段階までしかTNFAを構成しなかったので、抽象化が必要となる様先の例を意図的に若干修正して、第77図乃至第79図にC-TNFA2TNFAでAbstraction of TNFAを適時コールした場合の処理の経過の一部を示す。ここでは、klには具体的な値は与えないものとする。

## 《パラメトリック解析結果》

前記文献に述べられているアルゴリズムを用いて、遷移を2回以上通らない制約で、探索深さ最大値16として、以下の検証性質

EF (<displayend>( (AF(<renderbegin>true))  
and ((AF(<renderend>true)) AU (<displaybegin>true))))

に関してパラメタ条件を導出すると、以下の条件、

0 <= kd && 0 <= kr2 && 0 <= kr1 && 0 <= kb3 && 0 <= kb2 && 0 <= kb1  
&& 0 <= kloop1 && 0 <= k5 && 4 <= k4 && 0 <= k3 && 0 <= k1 && 0 <= k2 && 12 <= kb1+9kb2+kb3 && 4 <= kl)

を得る。これと先に求めた、遷移を2回以上通らないという制約から得られたパラメータ条件

0<=k1+k2+kloop1<=7                      &&                      5<=kb1+kb2+kb3+kl<=8                      &&  
7<=kb1+3kb2+kb3+kl<=11 && 1<=kl<=11 && 1<=k1+k2+k3+kloop1+kl<=6  
&& 3<=k4<=11 && 5<=kb1+kb2+kb3+kl<=11 && 6<=kb1+4kb2+kb3+3kr1<=12

$\&\& 1 \leq 3kr2 + kl \leq 8 \ \&\& 9 \leq kb1 + 5kb2 + kb3 + kl \leq 12 \ \&\& kr1 = kb2 - 1 \ \&\& 8 \leq kb1 + 5kb2 + kb3 \leq 24 \ \&\& 1 \leq kl + 6kd \leq 19$

との論理積を取って、得られたパラメータ条件に対して、目的関数を各パラメータ値の合計として、これを最小化するという線形計画問題をフリーソフト LP\_SOLVE にて解かせると、第 80 図の解を得る。

第 80 図の結果では、バスアクセス終了通知が 0 サイクルで行われなければならない、また、描画サイクル  $kr2$  と表示サイクル  $kd$  が 0 であるので、意味のある解とは言えないので、 $kb1=2, kb2=1, kb3=1$  の制約を追加し、第 81 図の解を得た。

第 81 図の結果でも、描画サイクル  $kr2$  と表示サイクル  $kd$  が 0 であるので、意味のある解とは言えないので、 $kb1=2, kb2=1, kb3=1$  の制約に加えてさらに、 $kr1$  以外のパラメータ変数は 1 以上という制約を追加し、第 82 図の解を得た。以後、このパラメータ値を採用して議論を進める。

#### 《ハード合成》

ハードウェア合成処理についてその概要を先ず説明する。(1)～(11)の方針にてハードウェアの生成が行なわれる。

(1) 事前に Register クラスに登録された通信メソッドの種類からバスコマンドの割り当てを行う。

(2) ユーザ情報として、どのデバイスにどの共有レジスタが存在するかを受付け、各デバイス内のレジスタ数に対応するデバイス内アドレスを割り当てる。

(3) 共有レジスタが割付られたデバイス (run()メソッド) の数を取得し、共有レジスタが割り当てられたデバイスにグローバルなアドレスを割り当てる。

(4) インライニング (展開) を実施しなかった通信メソッド以外のメソッドのインライニングを行う。(通信メソッドと指定したメソッド

以外のメソッドは事前にインライニングされている事に注意。)

(5) パラメトリック解析結果から得られた、Basic Block の実行サイクルを各デバイスの合成用 C F G に反映する。

5 (6) 各デバイス夫々の C F G を変形し、通信メソッドとそれ以外のコントロールフロー部が互いに通信するモデルに変換する。

(7) 前記(6)で得られた変換後の C F G 内の通信メソッドに対応する部分に、バスコマンド生成記述を挿入する。通信メソッドは、グローバルアドレス、共有レジスタアドレス、バスコマンドを出力し、データ読み出し・書き込みの何れかを実行する記述からなる C F G となり、  
10 それらの信号生成・受理に要するサイクルは、パラメトリック解析の結果により決定される。

(8) 共有レジスタアドレスから、各デバイス内のレジスタの配列インデックスを生成するアドレスデコーダとグローバル・アドレスを識別するアドレスデコーダを共有レジスタが割り当てられたデバイス内に  
15 生成し、出力トライステートを共有レジスタの出力に接続し、トライステート・イネーブル信号として、各デバイスからのデータ入カステージを表す信号の論理和を用いる。共有レジスタの入力に関しては、取り込み動作が起こらない限り内部変数は前の値を保持する為バスから信号を接続すれば良よく、各デバイスからのデータ入カステージを表す信号の論理和を元にデータ取り込み判定を行う。但し、これらは C F G と  
20 して表現する。(自デバイス内に共有レジスタがあったとしても、この方式に従って、アクセスが行われているものとする。)

(9) 変形等で得られた各 C F G の信号通信関係(入力、出力)を識別し(バスは入出力)、夫々をモジュールとして識別する。

25 (10) 各 C F G を「サイクルアキュレートなプログラム記述からのハード合成」に従って、H D L へと変換する。

(11) バスに対してリピータ回路をHDLの形式で挿入する。

前記バスコマンドの割り当てに関し、本例で登録したバスアクセス・メソッドは、sync\_read、sync\_write、sync\_burst\_read、sync\_burst\_write、endBurstAccessであるが、各run()メソッドのCFGを解析すると、実際に用いられているバスアクセスメソッドは、

5 sync\_burst\_read、sync\_burst\_write、endBurstAccessであるため、バスコマンドを

```
sync_burst_read 2'b00
sync_burst_write 2'b01
10 endBurstAccess 2'b10
NOP 2'b11
```

のように割り当てる。

上記アドレス割り当てに関しては、共有レジスタは Unified Memory にのみ割り付けられ、Unified Memory の仕様から割り付けられた共有

15 レジスタは

```
mem_con_reg.current_value[0]: コマンドフラグ0、
mem_con_reg.current_value[1]: 描画コマンド0、
mem_con_reg.current_value[2]: コマンドフラグ1、
mem_con_reg.current_value[3]: 描画コマンド1、
20 mem_con_reg.current_value[4]: 描画ソース・データ、及び描画後の表示用データ、
mem_con_reg.current_value[5]: 描画ソース・データ、及び描画後の表示用データ、
mem_con_reg.current_value[6]: 描画ソース・データ、及び描画後の表示用データ、
25 mem_con_reg.current_value[7]: 描画ソース・データ、及び描画後の表示用データ、
```

示用データ、

mem\_con\_reg.current\_value[8] : 描画ソース・データ、及び描画後の表示用データ、

- 5 mem\_con\_reg.current\_value[9] : 描画ソース・データ、及び描画後の表示用データ、となる。尚、ユーザ情報から各レジスタのビット幅が指定されているものとし、それぞれ32ビット (unsigned int) であるとする。

バスアクセス・メソッドはレジスタ内のバイトアクセス等を行わないので、各レジスタのアドレスを

- 10 mem\_con\_reg.current\_value[0] : 4'b0000、  
mem\_con\_reg.current\_value[1] : 4'b0001、  
mem\_con\_reg.current\_value[2] : 4'b0010、  
mem\_con\_reg.current\_value[3] : 4'b0011、  
mem\_con\_reg.current\_value[4] : 4'b0100、  
15 mem\_con\_reg.current\_value[5] : 4'b0101、  
mem\_con\_reg.current\_value[6] : 4'b0110、  
mem\_con\_reg.current\_value[7] : 4'b0111、  
mem\_con\_reg.current\_value[8] : 4'b1000、  
mem\_con\_reg.current\_value[9] : 4'b1001、のように決定する。

- 20 また、アドレス割り当てに関し、共有レジスタが割り当てられたデバイスは Unified Memory しか存在しないため、グローバルアドレスの割付を行わない。

尚、本例では1つのデバイスのみにより共有レジスタの割り当てを行っているが、以降の説明に於いてこれは一般性を欠くものではない。それは、  
25 以降に処理手順を示す事で明らかとなる。

前記 BasicBlock への実行サイクルの割り当てに関しては、ここでは、

Command Interface のみ取り上げて説明を行う。特に、先に求めたパラメータ値から、 $k1 = 4$ ,  $kb1 = 2$ ,  $kb2 = 1$ ,  $kb3 = 1$ ,  $k1 = k2 = k3 = 1$ 、として説明を行う。パラメータ条件から、各 BasicBlock の下に割り当てられたサイクル数のクロック境界を挿入する。ここでは、バスアクセス・メソッドを扱う対象としていない事に注意。更に、与えたパラメータで、例を示しているが一般性を失わない事に注意。次ページに変形後の C F G を示す。尚、 $k1$  の値は、直接各デバイスのハード合成用 C F G には影響を与えない事にも注意。これは、固定優先度スケジューラにのみ影響する。第 8 3 図には BasicBlock への実行サイクルの割り当ての例が示される。

固定優先度スケジューラの修正に関しては、既に得られている F S M (Finite State Machine) の遷移サイクルを  $k1$  に変更する、即ち、クロック境界を各遷移枝に 4 つ付加すれば良い。ここからの H D L 生成は、各状態からの遷移を分岐ノードに表現し直し、かつ各状態での信号代入はその状態への遷移枝上での最後のクロック境界直下に BasicBlock を設け、そこにレジスタ代入文を記述する形式で置き換える事で、この F M S 自体を C F G と見做す事で、「サイクルアキュレートな記述からの H D L 生成」を用いる事で可能である。第 8 4 図に示す固定優先度スケジューラの一部を用いて変形後の C F G が第 8 5 図に示される。

C F G の変形に関しては、元の C F G からバスアクセス・メソッドの括りだしを行い、バスアクセス・メソッドと元の C F G との通信ノードを設る。具体的には、下記を行う。

1) ハード合成用の C F G 生成段階で、クロック境界を含むバスアクセス・メソッドのクロックをバスアクセス・メソッドを表すノードの直下にクロック境界を追加したが、これを削除する。

2) バスアクセス・メソッドを表すノードを下記処理を行う Basic



Block に置き換える。

① 出力信号 `start_comm` に 1 を代入する。

② バスコマンドを出力信号 `bus_cmd` に代入する。バースト転送の場合、初期サイクルでは、

5     <1> 出力信号 `init` の 0 を代入

      <2> アクセスするデバイスのアドレスと共有レジスタのアドレスを連結して出力信号 `address` に代入

      <3> クロック境界挿入  $\times n \text{ b f}$

      <4> リードメソッドなら、代入すべき変数に入力信号 `data_in` を代入

10    <5> ライトメソッドなら、出力すべき値または変数を出力信号 `data_out` に代入

      <6> クロック境界  $\times m \text{ b f}$ 、を行う。ここで、 $n \text{ b f} + m \text{ b f} = k \text{ b } 1$  である。この例では、 $n \text{ b f} = m \text{ b f} = 1$  とする。

バースト転送の場合、転送終了では、

15    <1> クロック境界挿入  $\times n$ 、を行う。ここで、 $n = k \text{ b } 3$  である。ここでは、 $n = 1$  とする。

バースト転送の場合、それ以外では、

      <1> 出力信号 `init` に 1 を代入

      <2> アクセスするデバイスのアドレスと共有レジスタのアドレスを連結して出力信号 `address` に代入

20    <3> クロック境界挿入  $\times n \text{ b}$

      <4> リードメソッドなら、代入すべき変数に入力信号 `data_in` を代入

      <5> ライトメソッドなら、出力すべき値または変数を出力信号 `data_out` に代入

25    <6> クロック境界  $\times m \text{ b}$  を行う。ここで、 $n \text{ b} + m \text{ b} = k \text{ b } 2$  である。この例では、 $n \text{ b} = 0$ 、 $m \text{ b} = 1$  とする。

シングル転送の場合、

<1> アクセスするデバイスのアドレスと共有レジスタのアドレスを連結して出力信号 address に代入

<2> クロック境界挿入  $\times n s$

5 <3> リードメソッドなら、代入すべき変数に入力信号 data\_in を代入

<4> ライトメソッドなら、出力すべき値または変数を出力信号 data\_out に代入

<5> クロック境界挿入  $\times m s$ 、を行う。ここで、 $n s + m s = k s$  である。

10 ③ 出力信号 start\_comm に 0 を代入する。

3) バスアクセスメソッドを表す孤立ノードを作成し、置き換えて生成した Basic Block との通信ノードを設ける。通信ノードとして、

start\_comm : Basic Block  $\rightarrow$  孤立ノード

bus\_cmd : Basic Block  $\rightarrow$  孤立ノード

15 address : Basic Block  $\rightarrow$  孤立ノード

data\_in : 孤立ノード  $\rightarrow$  Basic Block

data\_out : Basic Block  $\rightarrow$  孤立ノード、を設ける。

第 86 図及び第 87 図には変形後の Command Interface の C F G の一部が示される。第 87 図において、クロック境界の傍らに記載された変数  
20 数はパラメータであり、その数だけクロック境界を生成することを意味する。

孤立ノードへの C F G の割り当てに関しては、第 88 図の C F G を自動生成する。第 88 図において、特に、AD\_BUS はアドレスバスを、D\_BUS はデータバスを表し、data\_in\_en\_i はデータ入力ステージを、  
25 data\_out\_en\_i はデータ出力ステージを表す。ここで、i は各デバイスのインデックスを表す。また、アドレスバスのバス幅は、割り当てたバ

5 スコマンドのビット幅とアドレスのビット幅の合計とし、データバスはアクセスするレジスタのビット幅とする。また、クロック境界の傍にかかれた変数はパラメータであり、その数だけクロック境界を生成する事を意味する。尚、各変数の値は、C F Gの変形過程で用いた変数の値と等しい。

共有レジスタに関しては、第 8 9 図の記述をインライン展開したもの  
10 に対応する C F G の生成を行えばよい。但し、入出力変数である AD\_BUS、D\_BUS の入力、出力への分離は行わず、また C F G 上での最適化に於いては、AD\_BUS、D\_BUS に対する変数最適化は実施しないものとする。第 8 9 図、第 9 0 図及び第 9 1 図には共有レジスタに関する擬似 C 記述である。サイクルアキュレートな記述からの H D L 生成に関しては本発明者による先の出願（特願 2 0 0 2 - 3 0 0 0 7 3）を参照することができる。

15 以上本発明者によってなされた発明を実施形態に基づいて具体的に説明したが、本発明はそれに限定されるものではなく、その要旨を逸脱しない範囲において種々変更可能であることは言うまでもない。

例えば、積オートマトンの構成に関しては、T N F A 積オートマトンを構成するとき、遷移枝の通過回数で Assume Guarantee Reasoning を実施したが、本来は、sync 動作の通過回数で行うべきである。理由は、  
20 sync 動作はバスアクセス動作を意味しており、並列に動いている T N F A が何回 sync 動作を実施したかは、その T N F A に対応するデバイスがバスアクセスを何回行ったかに対応するからであり、Refinement 及びパラメトリック・モデルチェッキングで得られるのは、遷移サイクルの条件のみならず、その検証性質を実施している間に各デバイスが  
25 高々どれだけバスアクセスを行うかの情報も得られるからである。但し、この手法であると、組み合わせ爆発を起し得るので、何らかの高速な枝

切りが必要となる。

- 積オートマトンの構成に関しては、今回は、パラメトリック・モデル  
チェックの停止性を吟味せず、バウンデッド・モデルチェック  
として、検証性質を満たす必要条件を求める事を行ったが、本来は十分  
5 を求める必要がある。その為、アルゴリズムの停止性に関する研究を更  
に行う必要がある。

バスモデルの拡張に関しては、単一双方向バスのみではなく、単方向  
バスや、ローカルバスを含むもの、またバス・ブリッジを介してバスが  
階層化されているような複数バスシステムを扱うようにしてもよい。

- 10 また、水平帰線期間のみを扱ったが、本来は垂直帰線期間もモデル化  
して、1 フレームを表示している間に最低 1 フレーム分の描画が終了す  
るための十分条件を求める必要がある。

#### 産業上の利用可能性

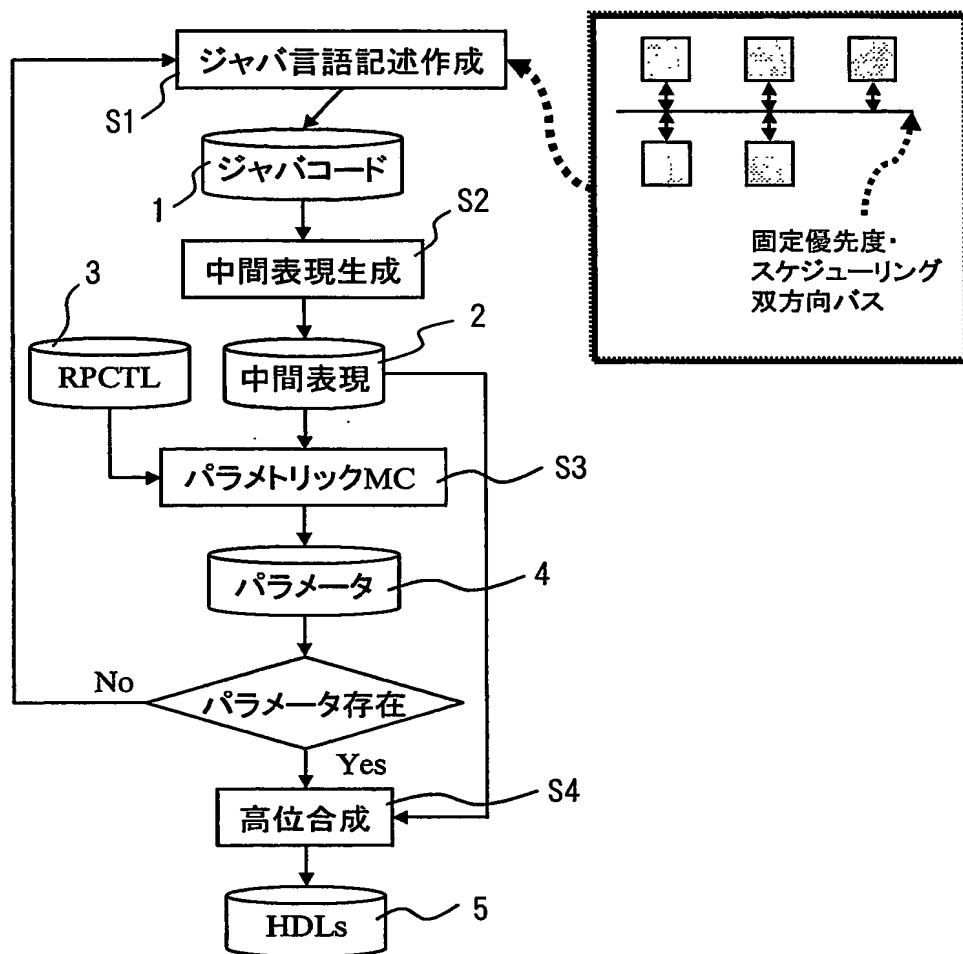
- 15 本発明は、並列動作を記述できる言語からデジタル回路のハードウ  
ェア合成を行うデータ処理システムなど広く適用することができる。

## 請 求 の 範 囲

1. 並列動作の記述が可能なプログラム言語を用いて複数のデバイスを定義したプログラム記述を入力し、入力したプログラム記述を中間表現に変換し、この中間表現に対し、実時間制約を満足するパラメータを生成し、生成したパラメータに基づいてハードウェア記述言語による回路記述を合成することを特徴とするシステム開発方法。
- 5
2. 前記中間表現は、コンカレントなコントロールフローフラグ、コンカレントなパラメータ付き時間オートマトン、又はパラメータ付き時間オートマトンであることを特徴とする請求項 1 記載のシステム開発方法。
- 10
3. 前記パラメータ生成に、パラメトリック・モデルチェッキングを行うことを特徴とする請求項 2 記載のシステム開発方法。
4. 前記実時間制約は R P C T L で与えられることを特徴とする請求項 15 3 記載のシステム開発方法。
5. 前記プログラム記述は `run` メソッドを用いてデバイスの定義を行い、バリア同期を用いてデバイスのクロック同期を定義することを特徴とする請求項 4 記載のシステム開発方法。
- 20
6. 並列動作の記述が可能なプログラム言語を用いて複数のデバイスを定義したプログラム記述を入力し、入力したプログラム記述を中間表現に変換し、この中間表現に対し、実時間制約を満足するパラメータを生成し、生成したパラメータに基づいてハードウェア記述言語による回路記述を合成することを特徴とするデータ処理システム。
7. 前記プログラム記述は `run` メソッドを用いてデバイスの定義を行い、バリア同期を用いてデバイスのクロック同期を定義することを特徴とする請求項 6 記載のデータ処理システム。
- 25

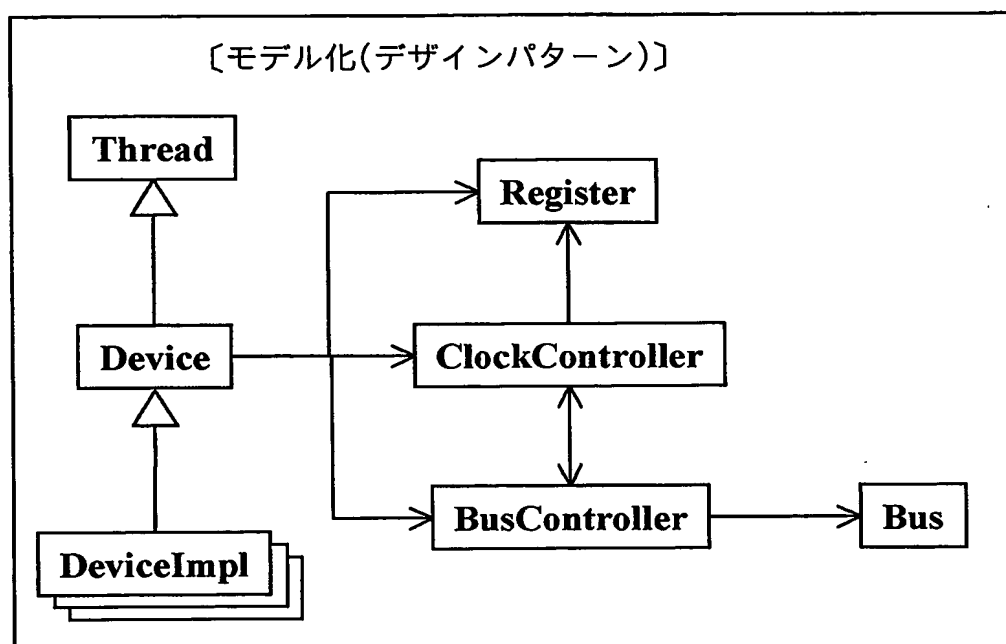
1 / 7 5

第1図



2 / 7 5

第 2 図



3 / 7 5

## 第 3 図

〔モデル化(クロック同期)〕

```
private void consume_1_clock() {
    /* 全デバイス数とこのmethodを実行したデバイスの数が等しいかチェック */
    if (++this.current_num == this.device_num) {
        this.current_num = 0;
        for (int i=0; i<this.device_num; i++) {
            /* レジスタ代入の実行 */
            registers[i].assignWriteValue();
        }
        /* バスロックを識別するフラグ変数(バスアクセス・フラグ)の初期化 */
        if (this.bc.getBusyCount() == 0) {
            this.bc.initLockDoneOnceFlag();
        }
        notifyAll();
    } else {
        try {
            wait();
        } catch (InterruptedException e) {
        }
    }
}
```

## 第 4 図

〔モデル化(クロック同期)〕

```
public void assignWriteValue() {
    /* 後述のsync_writeメソッド又は、sync_burst_writeメソッドにより
       レジスタへの書き込みが実行がなされたかを判定 */
    if (this.write_access) {
        /* 実際に書き込みを行ったレジスタ(配列index)への書き込みを
           実行 */
        this.current_value[this.update_index] = this.write_value;
        /* ライト・アクセス・フラグをリセット */
        this.write_access = false;
    }
}
```



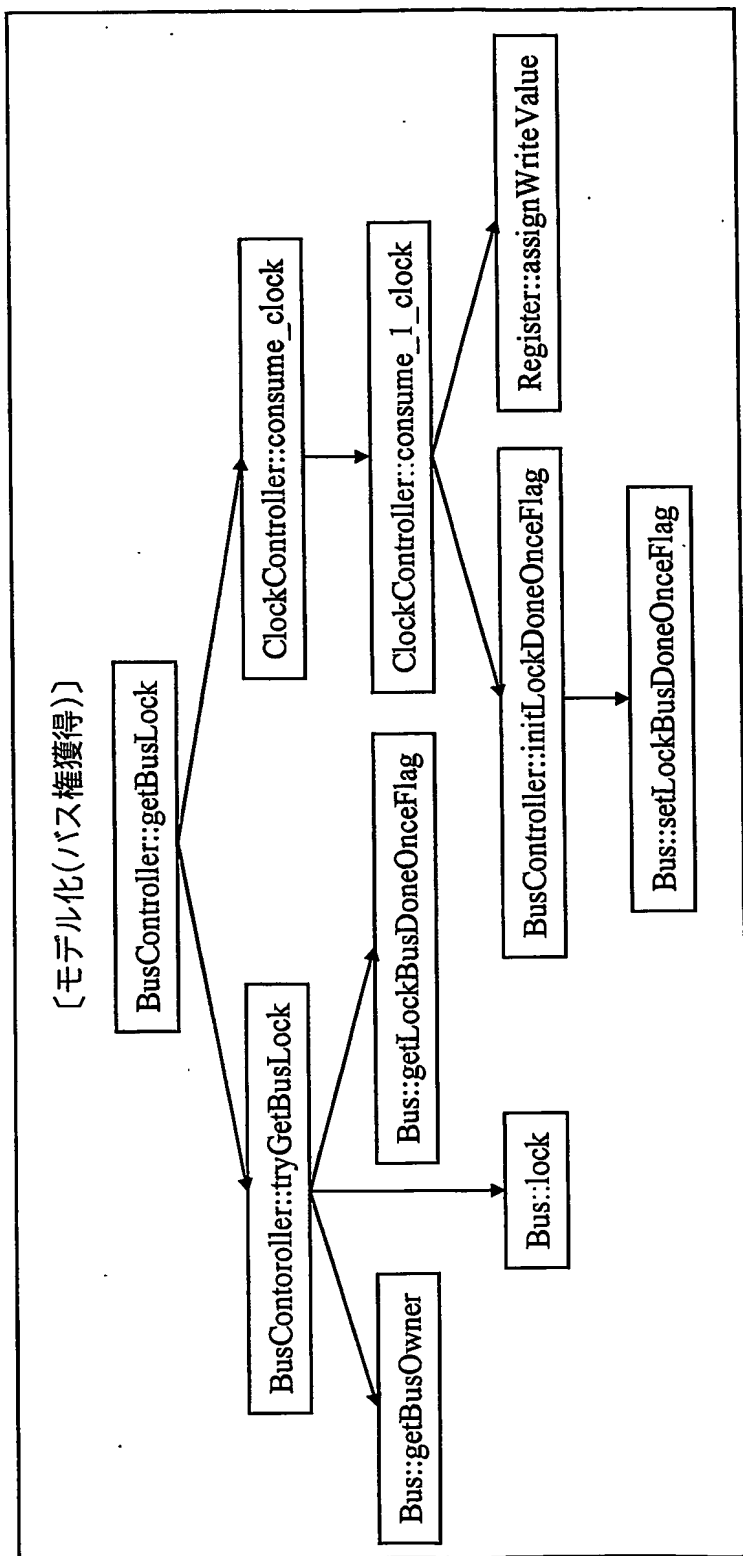
第5図

〔モデル化(バス権獲得)〕

method名	
getBusLock	tryGetBusLockがtrueを返せば、lockメソッドをコールする事でバス権を獲得し、falseの場合、consume_clockをコールし、clock_numの数だけクロック消費。
tryGetBusLock	getBusOwnerがnullで且つLockBusDoneOnceFlagがfalseなら、lockをコールし、getBusLockにtrueを返す。それ以外の場合はfalseを返す。
getBusOwner	現在バスをロックしているthreadのオブジェクトを返す。ロックしているオブジェクトが無ければnullを返す。
lock	現在、Bus classオブジェクト(lockメソッド)を実行しているThreadを、現在バスをロックしているオブジェクトとして登録する。
getBusDoneOnceFlag	バスアクセス・フラグの値を取得する。
consume_clock	consume_1_clockをコールする。
consume_1_clock	クロック同期メカニズムで既に説明済み。
initLockDoneOnceFlag	引数をfalseとしてsetLockBusDoneOnceFlagをコール。
setLockBusDoneOnceFlag	引数をバスアクセス・フラグに代入。
assignWriteValue	共有レジスタへのライトアクセスがあった場合、次のクロックへ進む前にレジスタ代入を実行する。

5 / 7 5

第6図



6 / 7 5.

第 7 図

## 〔モデル化(バス権獲得)〕

```
public void getBusLock(int clock_num) {  
    /* バス権が獲得出来たかチェック */  
    while (this.tryGetBusLock() == false) {  
        /* バス権が獲得出来なかったので、clock_numの数だけクロック消費 */  
        cc.consume_clock(clock_num);  
    }  
}
```

第 8 図

## 〔モデル化(バス権獲得)〕

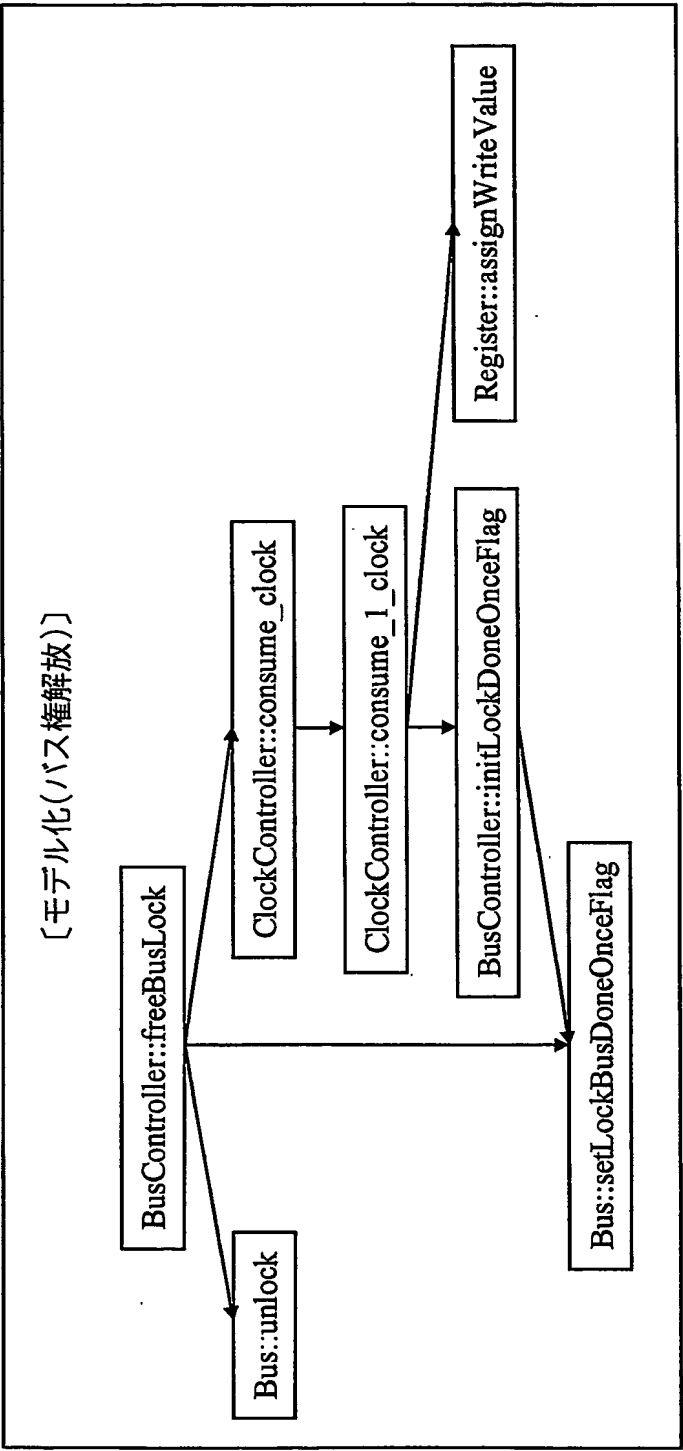
```
private synchronized boolean tryGetBusLock() {  
    /* 現在バスがロックされておらず、且つそのクロック内で一度もバスが  
       ロックされた事がない(即ちバスアクセス・フラグがfalse)か、をチェック */  
    if ((bus.getBusOwner() == null) && (bus.getLockDoneOnceFlag() == false)) {  
        /* バスをロック */  
        this.bus.lock();  
        /* ロック回数をインクリメント */  
        this.busycount++;  
        /* バス権が獲得できた事をtrueとして返す */  
        return true;  
    } else if (bus.getBusOwner() == Thread.currentThread()) {  
        /* 現在バスをロックしているスレッドによりさらにロック要求があった場合、  
           重ねてロックする。そのためにロック回数をインクリメント */  
        this.busycount++;  
        /* バス権が獲得できた事をtrueとして返す */  
        return true;  
    } else {  
        /* バス権が獲得出来なかった事をfalseとして返す */  
        return false;  
    }  
}
```

第9図

〔モデル化(バス権開放)〕

method名	
freeBusLock	unlockメソッドをコールする事でバス権を開放し、trueを引数として setLockBusDoneOnceFlagをコールし、consume_clockをコールし、 clock_numの数だけクロック消費。
unlock	現在バスをロックしているオブジェクトが、現在Bus classオブジェクト (lockメソッド)を実行しているThreadかを確認し、そうであれば、現在バスを ロックしているオブジェクトをnullとする。
consume_clock	consume_1_clockをコールする。
consume_1_clock	クロック同期メカニズムで既に説明済み。
initLockDoneOnceFlag	引数をfalseとしてsetLockBusDoneOnceFlagをコール。
setLockBusDoneOnceFlag	引数をバスアクセス・フラグに代入。
assignWriteValue	共有レジスタへのライトアクセスがあった場合、次のクロックへ進む前に レジスタ代入を実行する。

第 1 0 図



9 / 75

## 第 1 1 図

〔モデル化(バス権解放)〕

```
public void freeBusLock(int clock_num) {
    synchronized (this) {
        /* コールしたスレッドがバスをロックしているスレッドかチェック */
        if (this.bus.getBusOwner() == Thread.currentThread()) {
            /* ロック回数をデクリメント */
            this.busycount--;
            /* デクリメントした結果が0かどうかチェック */
            if (this.busycount == 0) {
                /* バスのロックを解放 */
                this.bus.unlock();
            }
            /* バスアクセス・フラグをtrueに設定 */
            this.bus.setLockDoneOnceFlag(true);
        }
    }
    /* clock_numの数だけクロック消費 */
    cc.consume_clock(clock_num);
}
```

## 第 1 2 図

〔モデル化(排他的同期リード)〕

```
public synchronized int sync_read(BusController bc,
                                   int index,
                                   int clock_num) {

    /* バス権獲得 */
    bc.getBusLock(clock_num);
    /* 指定したオブジェクト共有変数の値を読み込む */
    int read_value = this.current_value[index];
    /* バス権解放 */
    bc.freeBusLock(clock_num);
    /* 読み込んだ値を返す */
    return read_value;
}
```

〔sync\_read メソッド〕

10/75

## 第13図

〔モデル化(排他的同期リード)〕

```
public void run() {  
    Register other_r0 = (Register)super.access_registers.get(0);  
    int read_value;  
    while (true) {  
        this.do_something_w_or_wo_clock_boundary1();  
        read_value = other_r0.sync_read(super.bc0, 1);  
        this.do_something_w_or_wo_clock_boundary2();  
    }  
}
```

〔run()での記述例〕

## 第14図

〔モデル化(排他的同期リード)〕

```
public int sync_burst_read(BusController bc, int index,  
                           int clock_num) {  
    /* コールされる度にロック条件を満たすならロックを重ねる */  
    bc.getBusLock(clock_num);  
    /* 指定したオブジェクト共有変数の値を読み込む */  
    int read_value = this.current_value[index];  
    /* 読み込んだ値を返す */  
    return read_value;  
}
```

〔sync\_burst\_read メソッド〕

11 / 75

## 第15図

〔モデル化(排他的同期リード)〕

```
public void endBurstAccess(BusController bc, int clock_num) {  
    bc.freeBurstBusLock(clock_num);  
}
```

〔endBurstAccess メソッド〕

## 第16図

〔モデル化(排他的同期リード)〕

```
public void freeBurstBusLock(int clock_num) {  
    synchronized (this) {  
        /* コールしたスレッドがバスをロックしているスレッドかチェック */  
        if (this.bus.getBusOwner() == Thread.currentThread()) {  
            /* ロック回数を0に戻す */  
            this.busycount = 0;  
            /* バスのロックを解放 */  
            this.bus.unlock();  
        }  
        /* バスアクセス・フラグをtrueに設定 */  
        this.bus.setLockDoneOnceFlag(true);  
    }  
    /* clock_numの数だけクロック消費 */  
    cc.consume_clock(clock_num);  
}
```

〔 freeBurstBusLock メソッド〕



12/75

## 第17図

〔モデル化(排他的同期リード)〕

```
public void run() {
    Register other_r1 = (Register)super.access_registers.get(1);
    int read_value[10]
    while (true) {
        this.do_something_w_or_wo_clock_boundary1();
        /* バースト・リード(10回連続リード) */
        synchronized (this) {
            int i;
            for (i=0; i<10; i++) {
                read_value[i] = other_r1.sync_burst_read(super.bc, i, 1));
                super.cc.consume_clock(1);
                this.do_something_wo_clock_boundary1();
            }
            other_r1.sync_burst_read(super.bc, i, 1);
            other_r1.endBurstAccess(super.bc, 1);
            this.do_something_wo_clock_boundary2();
        }
        this.do_something_w_or_wo_clock_boundary2();
    }
}
```

〔run()での記述例〕

13/75

## 第18図

〔モデル化(排他的同期ライト)〕

```
public synchronized void sync_write(BusController bc,
                                     int write_value,
                                     int index, int clock_num) {

    /* バス権獲得 */
    bc.getBusLock(clock_num);
    /* 指定したオブジェクト共有変数への書き込み値を保持 */
    this.write_value = write_value;
    /* アクセスした配列を通知 */
    this.update_index = index;
    /* 共有変数への書き込みがあった事を通知
       (consume 1_clockにて次クロックへの遷移
       直前に共有変数に書き込みを実施) */
    this.write_access = true;
    /* バス権解放 */
    bc.freeBusLock(clock_num);
}
```

〔sync\_write メソッド〕

## 第19図

〔モデル化(排他的同期ライト)〕

```
public void run() {
    Register other_r0 = (Register)super.access_registers.get(0);
    int write_value;
    while (true) {
        this.do_something_w_or_wo_clock_boundary1();
        other_r0.sync_write(super.bc, write_value0, 1);
        this.do_something_w_or_wo_clock_boundary2();
    }
}
```

〔run()での記述例〕

## 第20図

〔モデル化(排他的同期ライイト)〕

```
public int sync_burst_read {(BusController bc, int write_value, int index, int clock_num) {  
    /* コールされる度にロック条件を満たすならロックを重ねる */  
    bc.getBusLock(clock_num);  
    /* 指定したオブジェクト共有変数への書き込み値を保持 */  
    this.write_value = write_value;  
    /* 共有変数への書き込みがあった事を通知  
       (consume_1_clockにて次クロックへの遷移  
       直前に共有変数に書き込みを実施) */  
    this.write_access = true;  
}
```

sync\_burst\_write メソッド

15 / 75

第 2 1 図

〔モデル化(排他的同期ライト)〕

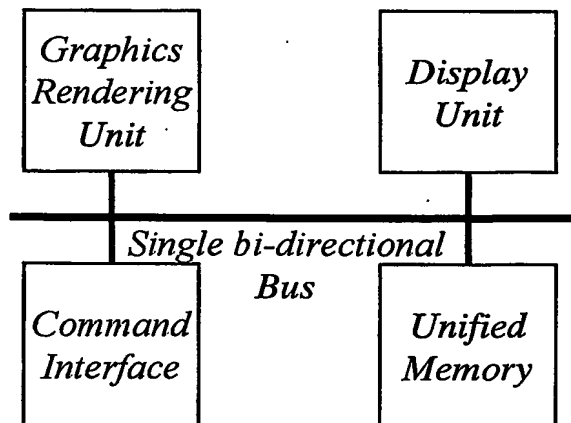
```

public void run() {
    Register other_r0 = (Register)super.access_registers.get(0);
    int write_value[10]
    while (true) {
        this.do_something_w_or_wo_clock_boundary1();
        /* バースト・ライト(10回連続ライト) */
        synchronized (this) {
            int i;
            for (i=0; i<10; i++) {
                other_r0.sync_burst_write(super.bc, write_value[i], 1));
                super.cc.consume_clock(1);
                this.do_something_wo_clock_boundary1();
            }
            other_r1.sync_burst_write(super.bc, write_value[i], 1);
            other_r1.endBurstAccess(super.bc, 1);
            this.do_something_wo_clock_boundary2();
        }
        this.do_something_w_or_wo_clock_boundary2();
    }
}

```

run()での記述例

第 2 2 図



16 / 75

第23図

[run() methodの実装例(Command Interface) ]

```

public void run() {
    Register mem_con_reg = (Register)super.access_registers.get(2);
    int[] drawing_commands = {100, 101, 102, 103, 104};
    int dcom_index = 0;
    int com_flag_0 = 0;
    int com_flag_1 = 0;
    while (true) {
        /* 外部入力信号write_req信号があり、且つwait_flag信号がfalseのとき
        コマンド受け付けを実行。それ以外の時は、単にクロックを消費するだけ。*/
        if (this.getWriteReqSignal() && (this.wait_flag == false)) {
            /* コマンドの受付を実行。*/
            this.input_command = drawing_commands[dcom_index++];
            /* 一定サイクル消費。ここでは3クロックとした。*/
            super.cc.consume_clock(3);
            /* wait_flag変数をtrueにセット*/
            this.wait_flag = true;
            /* write_req信号がありwait_flag変数がtrueなので、
            wait出力信号をtrueにセットし、1クロック消費。*/
            this.wait_output_signal = true;
            super.cc.consume_clock(1);
        }
    }
}

```

シミュレーション用に  
挿入した記述

17/75

## 第24図

[run() methodの実装例(Command Interface)]

```

/* コマンドフラグの値が0の場合に対応する描画コマンドの値を更新する。
   0,1共にコマンドフラグの値が0の場合は、0に対応するほうを優先する。
   0,1共にコマンドフラグの値が1の場合は、一定クロック待機して再度実行。*/
while (true) {
    synchronized (this) {
        /* コマンドフラグ0,1をバーストロードする。*/
        com_flag_0 = mem_con_reg.sync_burst_read(super.bc, 0, 1);
        super.cc.consume_clock(1); //クロック消費。
        com_flag_1 = mem_con_reg.sync_burst_read(super.bc, 2, 1);
        super.cc.consume_clock(1); //クロック消費。
        if (com_flag_0 == 0) {
            /* コマンドフラグ0の値が0の場合。または、コマンドフラグ0,1の値が共に0の場合。*/
            /* 描画コマンド0に入力されたコマンドを書き込む。*/
            mem_con_reg.sync_burst_write(super.bc, input_command, 1, 1);
            super.cc.consume_clock(1);
            /* コマンドフラグ0の値を1にする。*/
            mem_con_reg.sync_burst_write(super.bc, 1, 0, 1);
            /* バーストモードの終了(クロック消費含む)。*/
            mem_con_reg.endBurstAccess(super.bc, 1);
            break;
        }
    }
}

```

18 / 75

## 第25図

## 〔 run() methodの実装例 (Command Interface) 〕

```

    } else if (com_flag_1 == 0) {
        /* コマンドフラグ1の値が0の場合。*/
        /* 描画コマンド1に入力されたコマンドを書き込む。*/
        mem_con_reg.sync_burst_write(super.bc, input_command, 3, 1);
        super.cc.consume_clock(1);
        /* コマンドフラグ1の値を1にする。*/
        mem_con_reg.sync_burst_write(super.bc, 1, 2, 1);
        /* バーストモードの終了(クロック消費含む)。*/
        mem_con_reg.endBurstAccess(super.bc, 1);
        break;
    } else {
        /* 0,1共にコマンドフラグの値が1の場合。*/
        /* バーストモードの終了(クロック消費含む)。*/
        mem_con_reg.endBurstAccess(super.bc, 1);
        /* 一定クロック待機。ここでは3クロック待機させた。*/
        super.cc.consume_clock(3);
    }
} // end of synchronized
} // end of nested while-loop

```

## 第27図

## 〔 run() methodの実装例 (Unified Memory) 〕

```

public void run() {
    /* Unified Memoryへの書き込みを行うバス上のデバイス内の
       レジスタをインスタンス化 */
    // Graphics Rendering Unit 内のレジスタ群
    Register renderer_reg = (Register)super.access_registers.get(0);
    // Display Unit 内のレジスタ群
    Register display_reg = (Register)super.access_registers.get(1);
    // Command Interface 内のレジスタ群
    Register com_fetch_reg = (Register)super.access_registers.get(2);
    /* モデル簡略化の為、実際には何も行わない */
    while (true) {
        /* 1クロック消費。*/
        super.cc.consume_clock(1);
    }
}

```

19 / 75

第26図

[run() methodの実装例(Command Interface)]

```
/* wait_flag変数をfalseにセット。*/  
this.wait_flag = false;  
/* wait_flag変数がfalseになったので、wait出力信号をfalseに戻し、1クロック消費。*/  
this.wait_output_signal = false;  
super.cc.consume_clock(1);  
/* 描画コマンド配列を最後まで使用した場合はそのインデックスを最初に戻す。*/  
if (dcom_index == drawing_commands.length) {  
    dcom_index = 0;  
}  
} else {  
    super.cc.consume_clock(1);  
}  
} // end of while-loop  
}
```

シミュレーション用に  
挿入した記述



20 / 75

## 第28図

〔 run() methodの実装例 (Graphics Rendering Unit) 〕

```
public void run() {
    Register mem_con_reg = (Register)super.access_registers.get(2);
    int[] rendering_result = new int[6];
    int current_command = 0;
    int read_data = 0;
    int com_flag_0 = 0;
    int com_flag_1 = 0;
    while (true) {
        /* 待ち状態。*/
        super.cc.consume_clock(3);
        while (true) {
            synchronized (this) {
                /* コマンドフラグ0、1をバーストリードする。*/
                com_flag_0 = mem_con_reg.sync_burst_read(super.bc, 0, 1);
                super.cc.consume_clock(1); //クロック消費。
                com_flag_1 = mem_con_reg.sync_burst_read(super.bc, 2, 1);
                /* バーストモードの終了(クロック消費含む)。*/
                mem_con_reg.endBurstAccess(super.bc, 1);
            } // end of synchronized
        }
    }
}
```

## 第29図

[run() methodの実装例(Graphics Rendering Unit)]

```

if (com_flag_0 == 1) {
    /* コマンドフラグ0の値が0の場合。または、コマンドフラグ0,1の値が共に0の場合。*/
    synchronized(this) {
        /* 描画コマンド0の値を読み込む。*/
        current_command = mem_con_reg.sync_burst_read(super.bc, 1, 1);
        /* render_startをtrueに。*/
        this.render_start = true;
        super.cc.consume_clock(1);
        for (int i=0; i<3; i++) {
            /* データの読み出し。*/
            read_data = mem_con_reg.sync_burst_read(super.bc, i+4, 1);
            /* クロック消費。*/
            super.cc.consume_clock(1);
            /* レンダリング。*/
            rendering_result[i] = this.rendering(read_data, current_command);
        } // end of for-loop
        /* コマンドフラグ0の値を0にする。*/
        mem_con_reg.sync_burst_write(super.bc, 0, 0, 1);
        /* バーストモードの終了(クロック消費含む)。*/
        mem_con_reg.endBurstAccess(super.bc, 1);
    } // end of synchronized
}

```

22 / 75

## 第30図

〔 run() methodの実装例 (Graphics Rendering Unit) 〕

```
for (int i=3; i<6; i++) {  
    /* レンダリング。*/  
    rendering_result[i] = this.rendering(read_data, current_command);  
    /* クロック消費。*/  
    super.cc.consume_clock(1);  
} // end of for-loop  
break;  
} else if (com_flag_1 == 1) {  
    /* コマンドフラグ1の値が0の場合。*/  
    synchronized(this) {  
        /* 描画コマンド1の値を読み込む。*/  
        current_command = mem_con_reg.sync_burst_read(super.bc, 3, 1);  
        /* render_startをtrueに。*/  
        this.render_start = true;  
        super.cc.consume_clock(1);  
        for (int i=0; i<3; i++) {  
            /* データの読み出し。*/  
            read_data = mem_con_reg.sync_burst_read(super.bc, i+4, 1);  
            /* クロック消費。*/  
            super.cc.consume_clock(1);  
            /* レンダリング。*/  
            rendering_result[i] = this.rendering(read_data, current_command);  
        } // end of for-loop
```

23 / 75

## 第31図

## 〔 run() methodの実装例 (Graphics Rendering Unit) 〕

```

/* コマンドフラグ1の値を0にする。*/
mem_con_reg.sync_burst_write(super.bc, 0, 2, 1);
/* バーストモードの終了(クロック消費含む)。*/
mem_con_reg.endBurstAccess(super.bc, 1);
} // end of synchronized
for (int i=3; i<6; i++) {
/* レンダリング。*/
rendering_result[i] = this.rendering(read_data, current_command);
/* クロック消費。*/
super.cc.consume_clock(1);
} // end of for-loop
break;
} else {
/* 0,1共にコマンドフラグの値が1の場合。*/
/* 一定クロック待機。ここでは3クロック待機させた。*/
super.cc.consume_clock(3);
}
} // end of nested while-loop

```

## 第32図

## 〔 run() methodの実装例 (Graphics Rendering Unit) 〕

```

/* レンダリングされたデータをメモリに書き込む。*/
synchronized (this) {
for (int i=0; i<6; i++) {
if (i == 5) {
/* データの書き込み。*/
mem_con_reg.sync_burst_write(super.bc, rendering_result[i], i+4, 1);
/* バーストモードの終了(クロック消費含む)。*/
mem_con_reg.endBurstAccess(super.bc, 1);
} else {
/* データの書き込み。*/
mem_con_reg.sync_burst_write(super.bc, rendering_result[i], i+4, 1);
/* クロック消費。*/
super.cc.consume_clock(1);
}
} // end of for-loop
} // end of synchronized
/* レンダリング終了。*/
this.render_start = false;
} // end of while-loop
}

```

24 / 75

## 第33図

〔 run() methodの実装例 (Display Unit) 〕

```
public void run() {
    Register mem_con_reg = (Register)super.access_registers.get(2);
    int read_data = 0;
    while (true) {
        synchronized (this) {
            for (int i=0; i<6; i++) {
                if (i == 5) {
                    /* データの読み出し。*/
                    read_data = mem_con_reg.sync_burst_read(super.bc, i+4, 1);
                    /* バーストモードの終了(クロック消費含む)。*/
                    mem_con_reg.endBurstAccess(super.bc, 1);
                } else {
                    /* データの読み出し。*/
                    read_data = mem_con_reg.sync_burst_read(super.bc, i+4, 1);
                    /* 表示の為のデータロード開始。*/
                    this.display_start = true;
                    /* クロック消費。*/
                    super.cc.consume_clock(1);
                }
            } // end of for-loop
        } // end of synchronized
    }
}
```

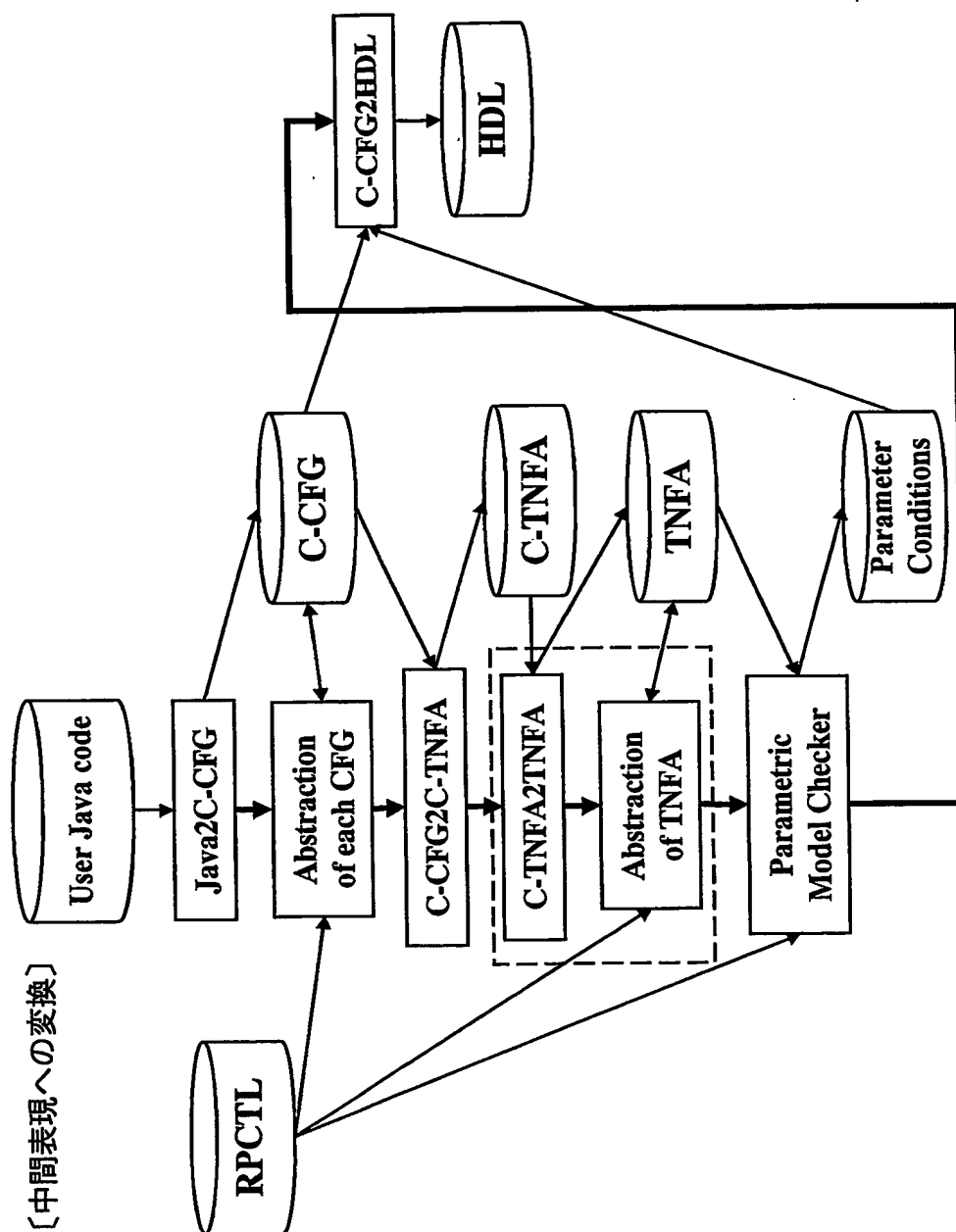
## 第34図

〔 run() methodの実装例 (Display Unit) 〕

```
/* 表示の為のデータロード終了。*/
this.display_start = false;
/* 表示。*/
for (int i=0; i<6; i++) {
    this.display(read_data);
}
/* 待ち。*/
super.cc.consume_clock(3);
}
}
```

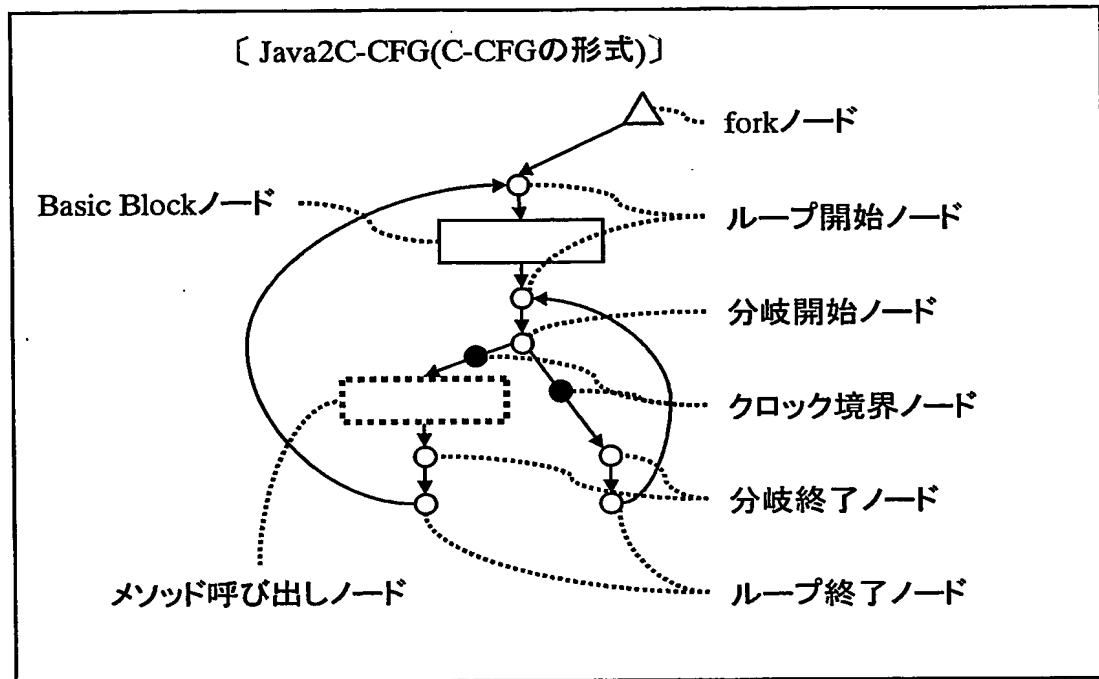
25 / 75

第35図

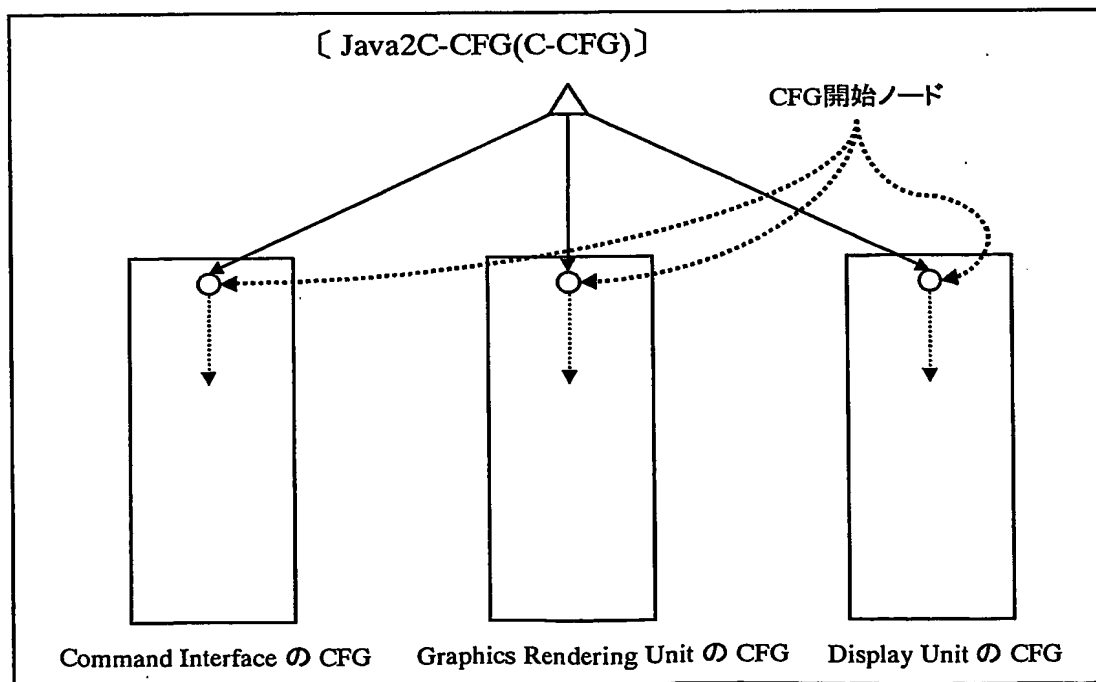


26 / 75

第36図

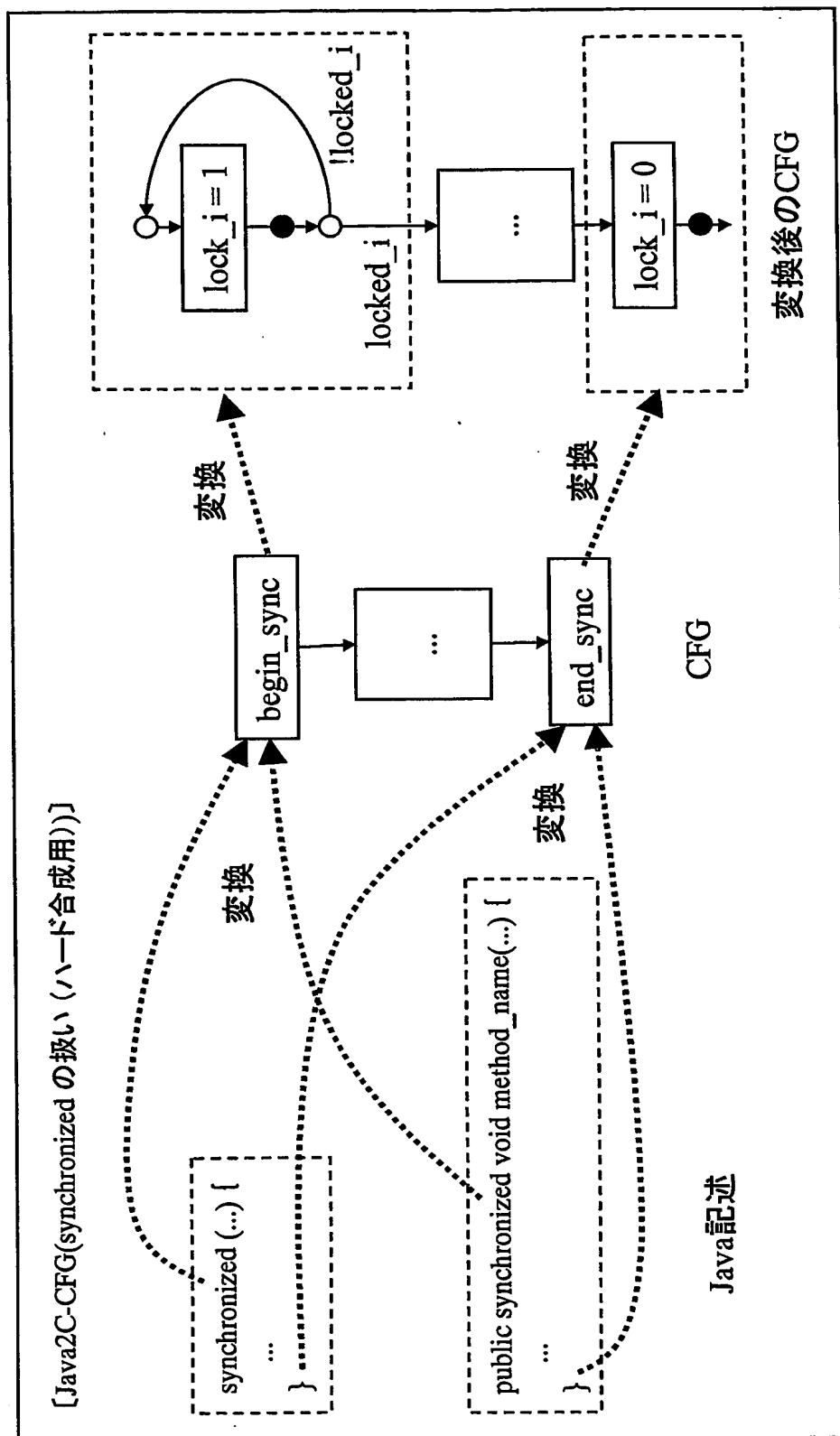


第48図



27 / 75

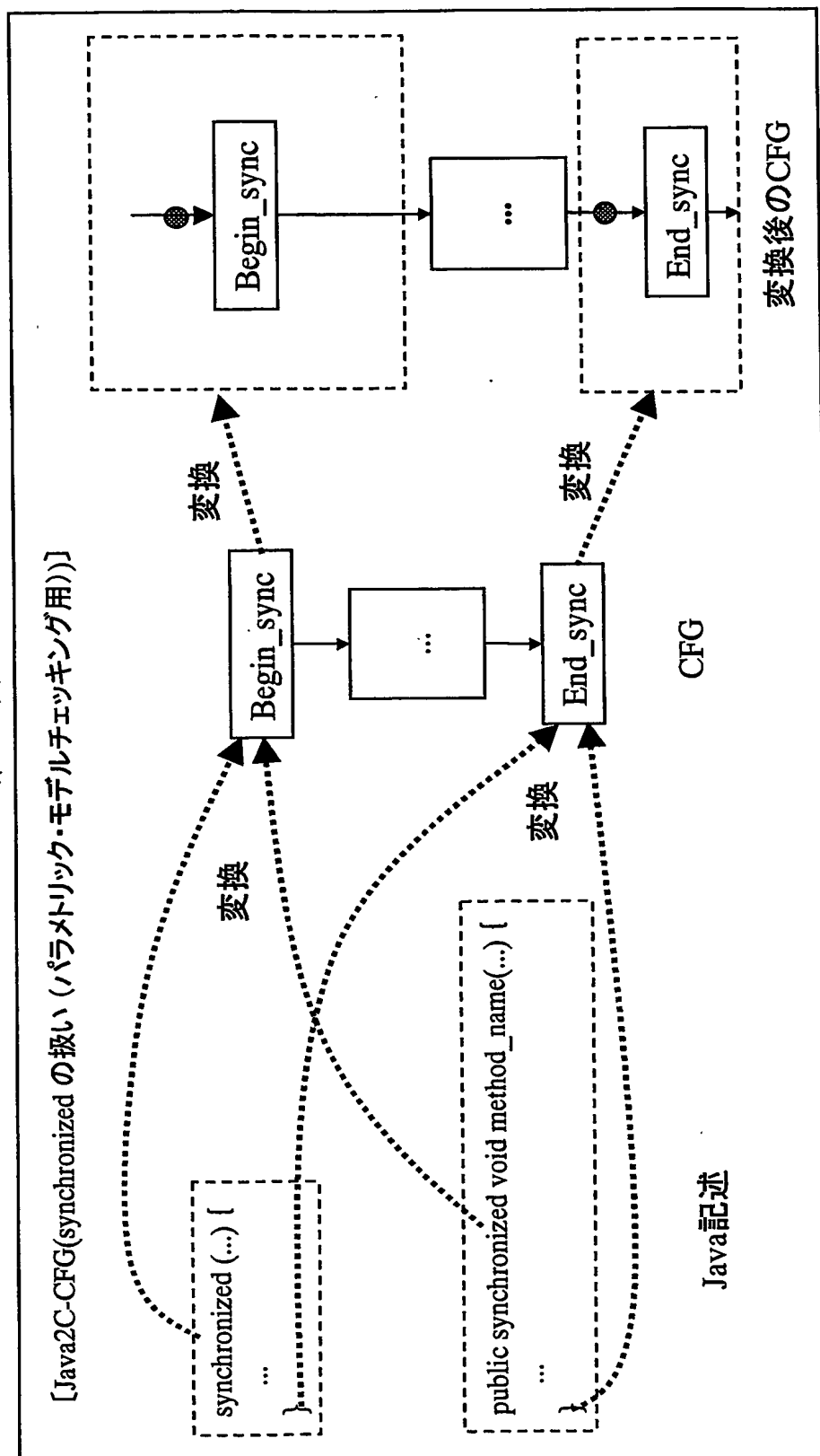
第37図





28/75

第38図

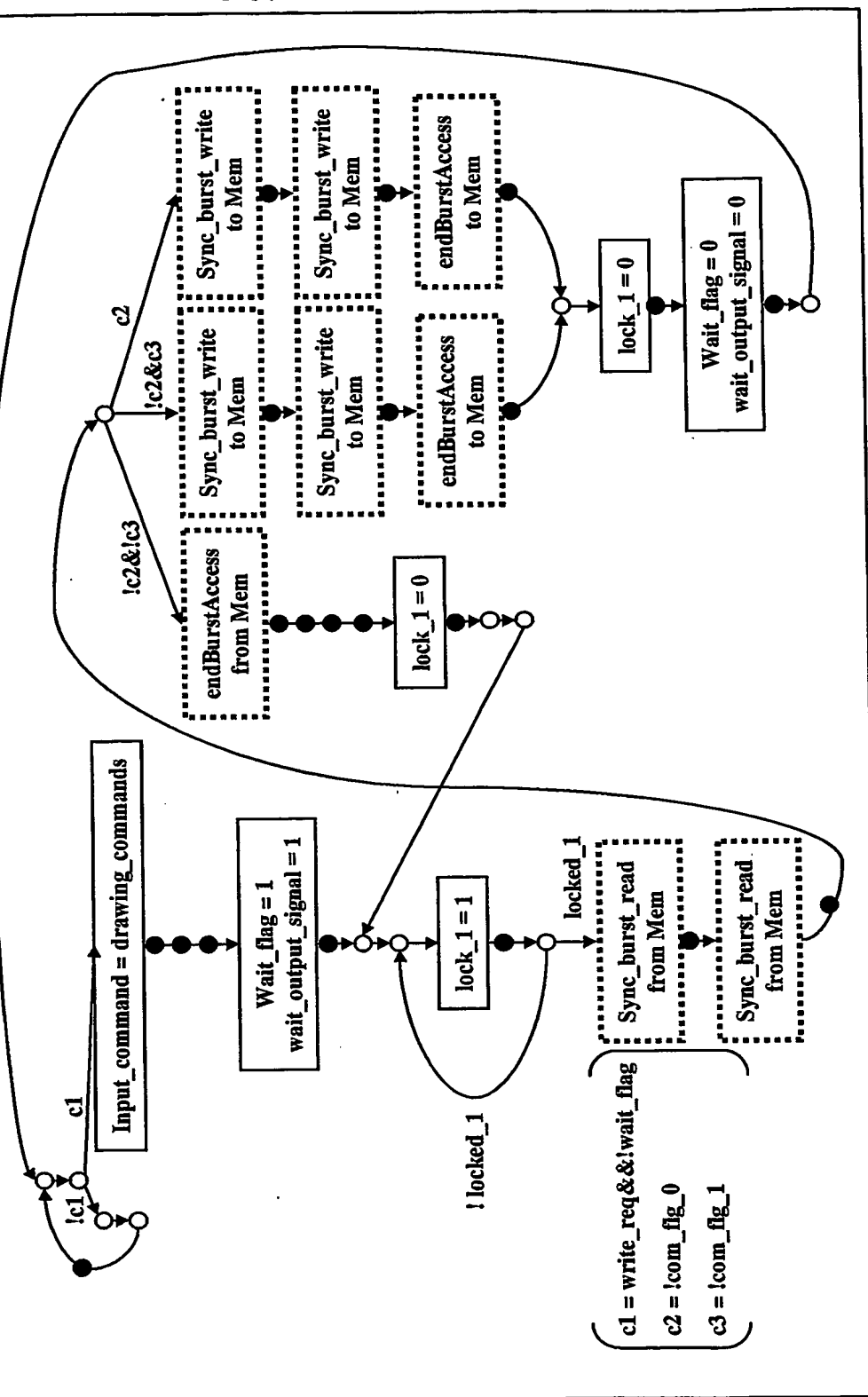




30/75

第40図

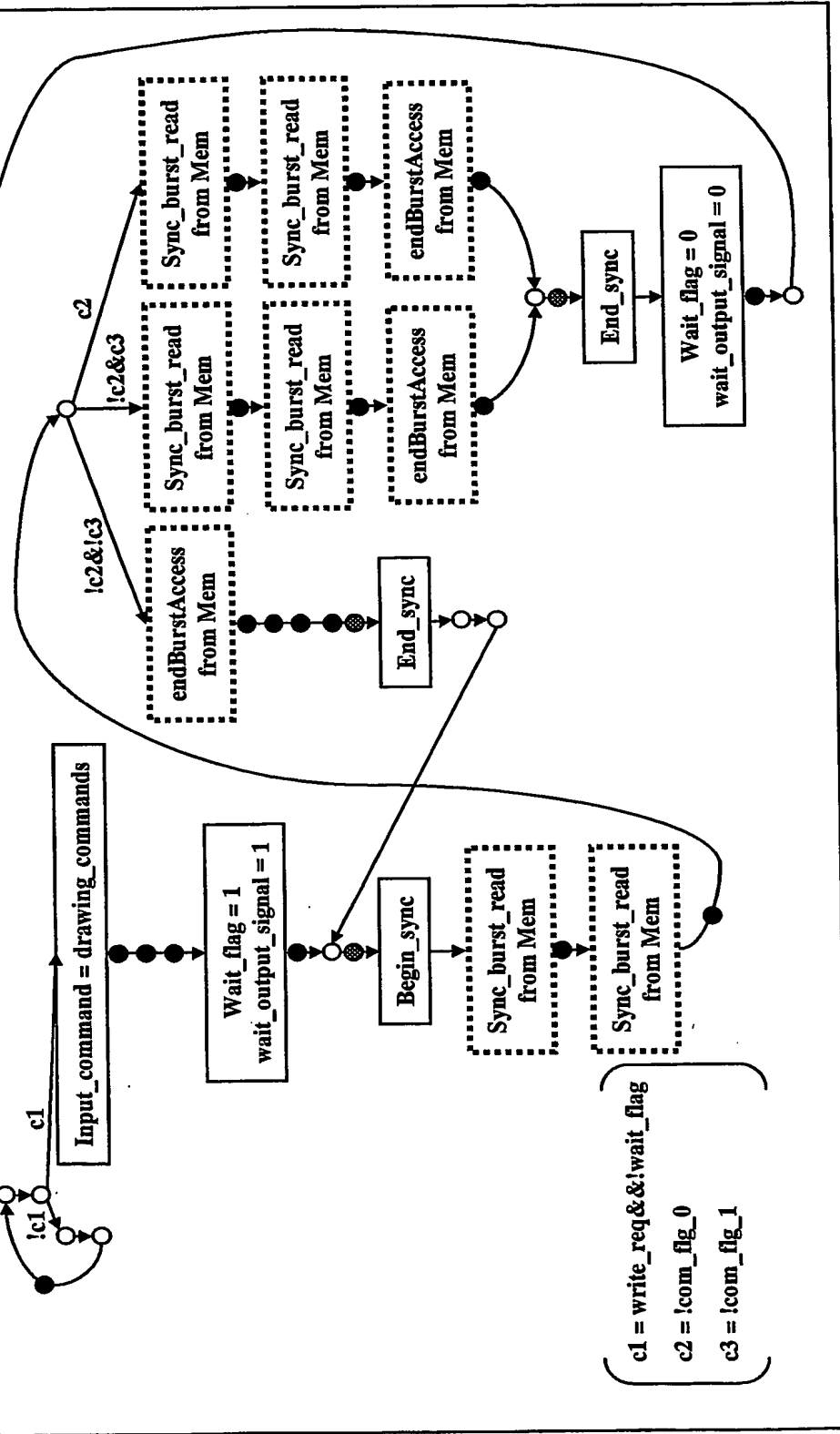
[Java2C-CFG(Command Interface (ハード合成用))]



31 / 75

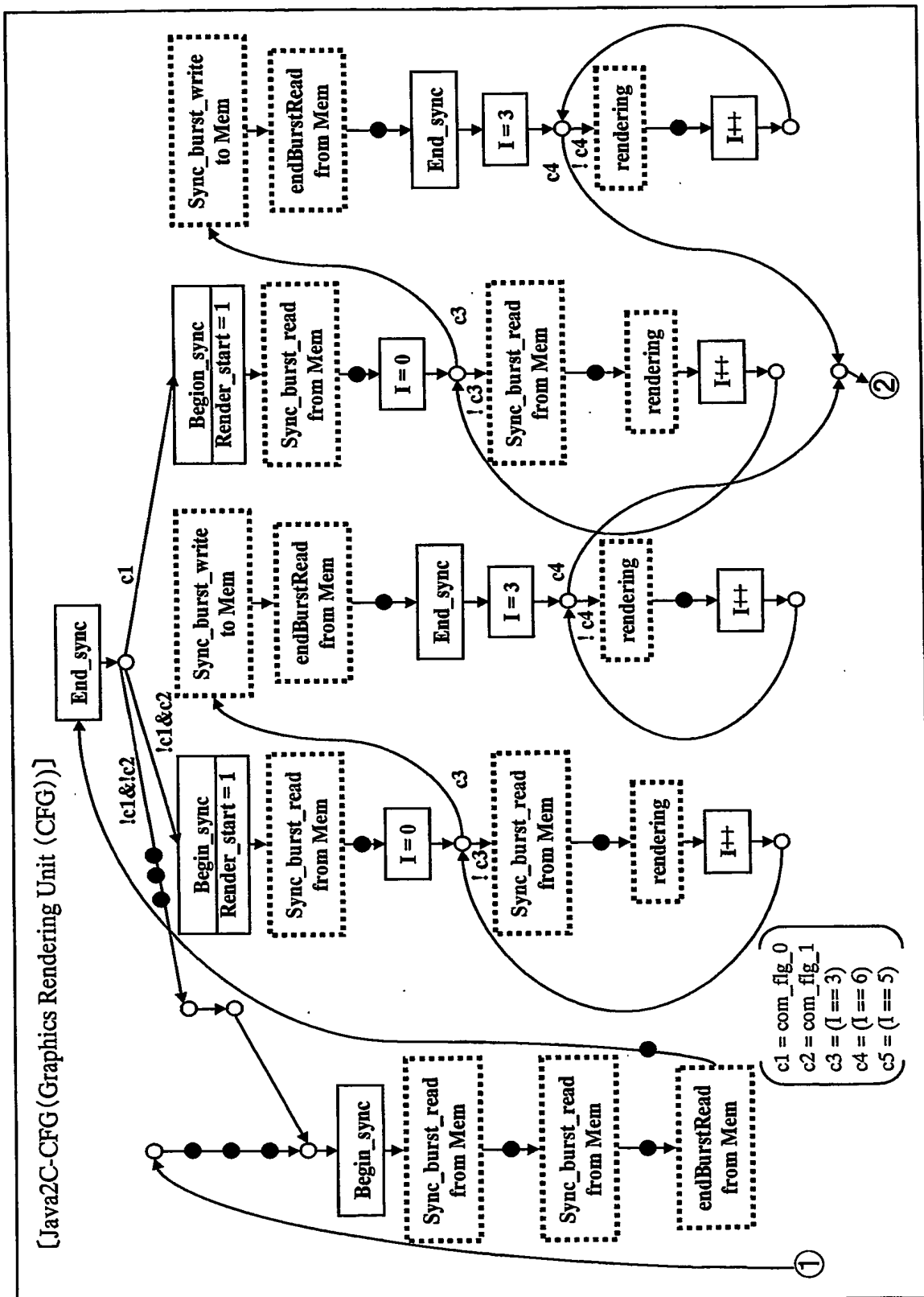
第41図

[Java2C-CFG(Command Interface (パラメトリック・モデルチェッキング用))]



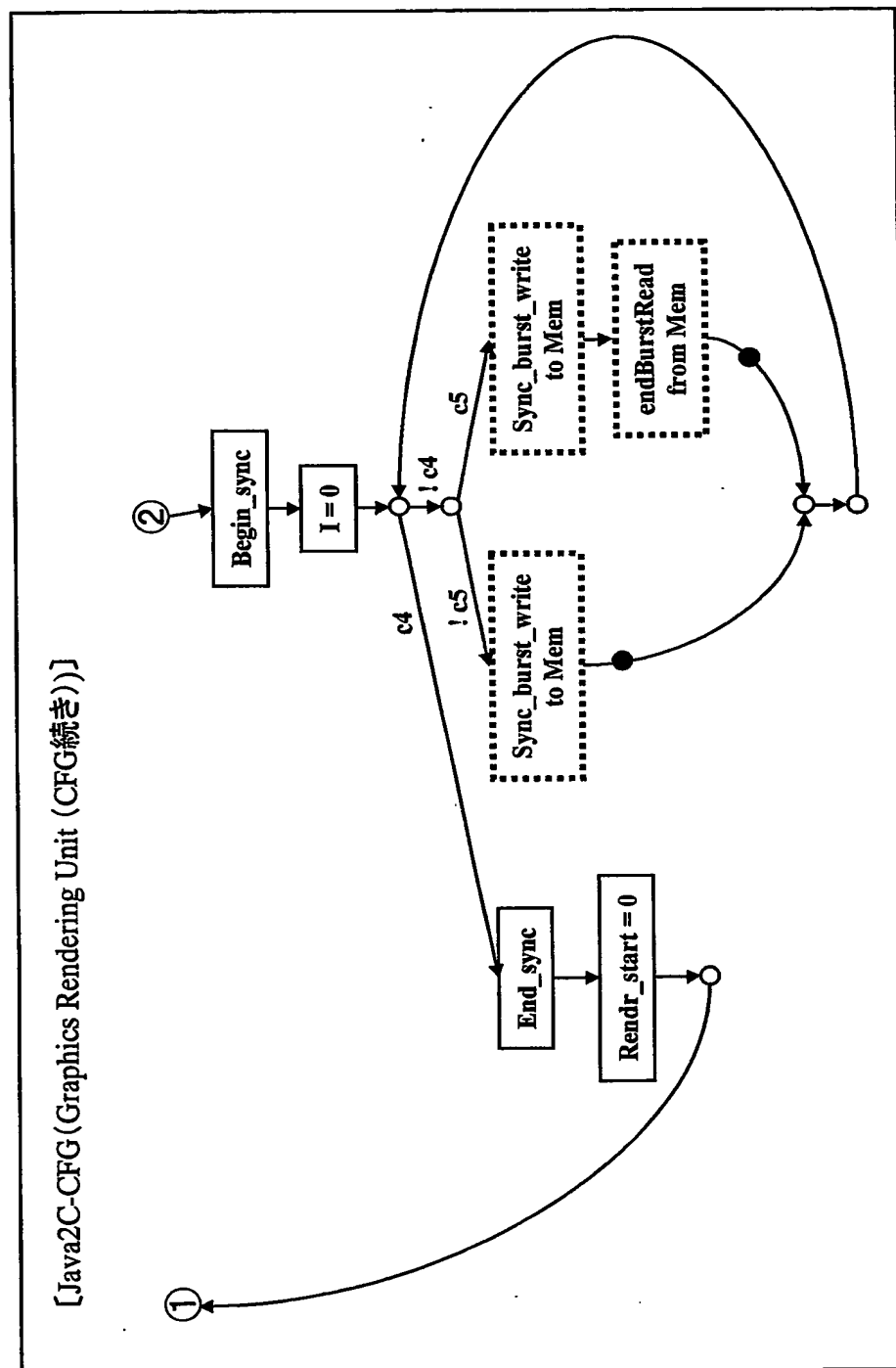
32 / 75

第42図



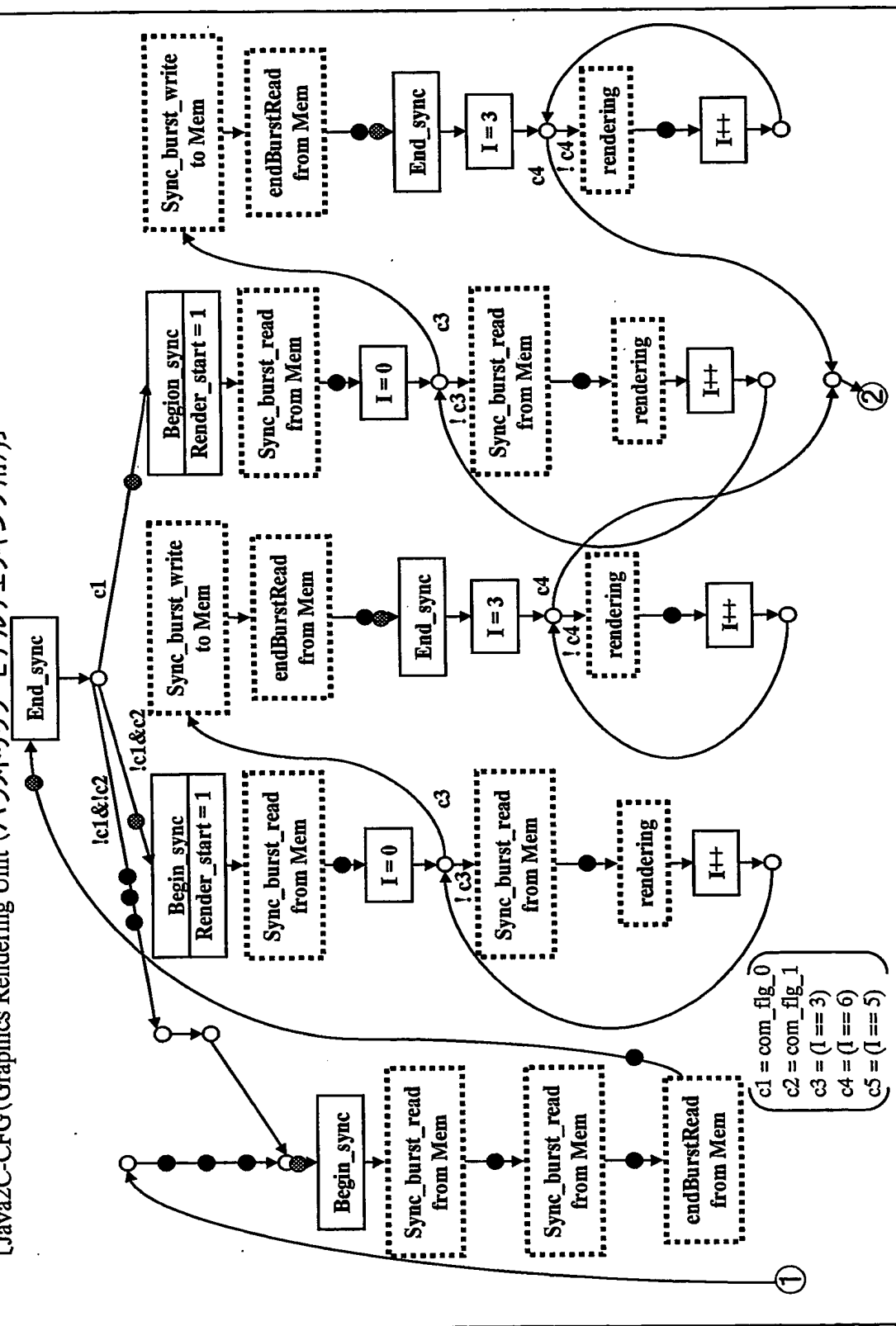
33 / 75

第43図



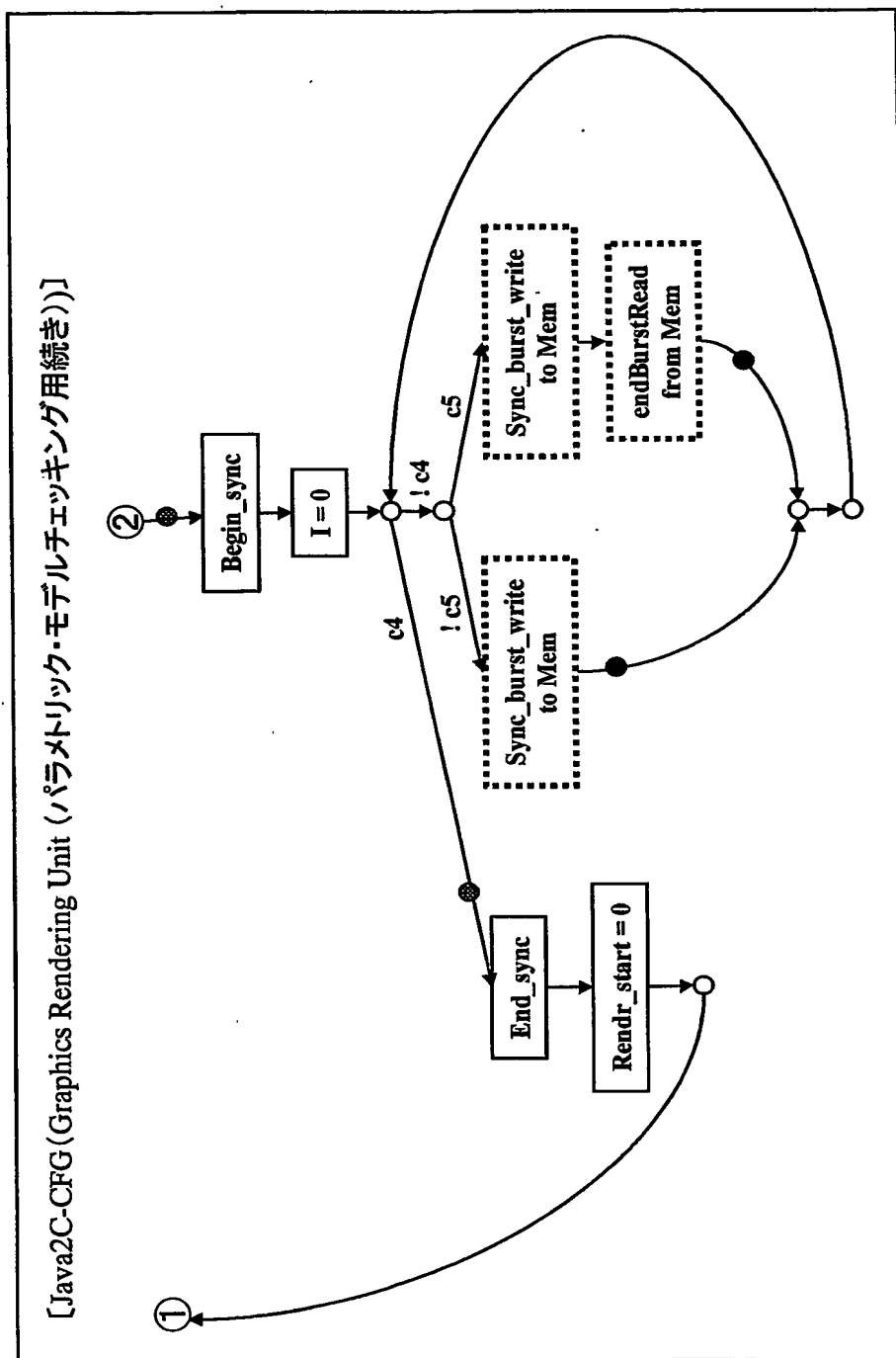
第44図

[Java2C-CFG (Graphics Rendering Unit (パラメトリック・モデルチェッキング用))]



35 / 75

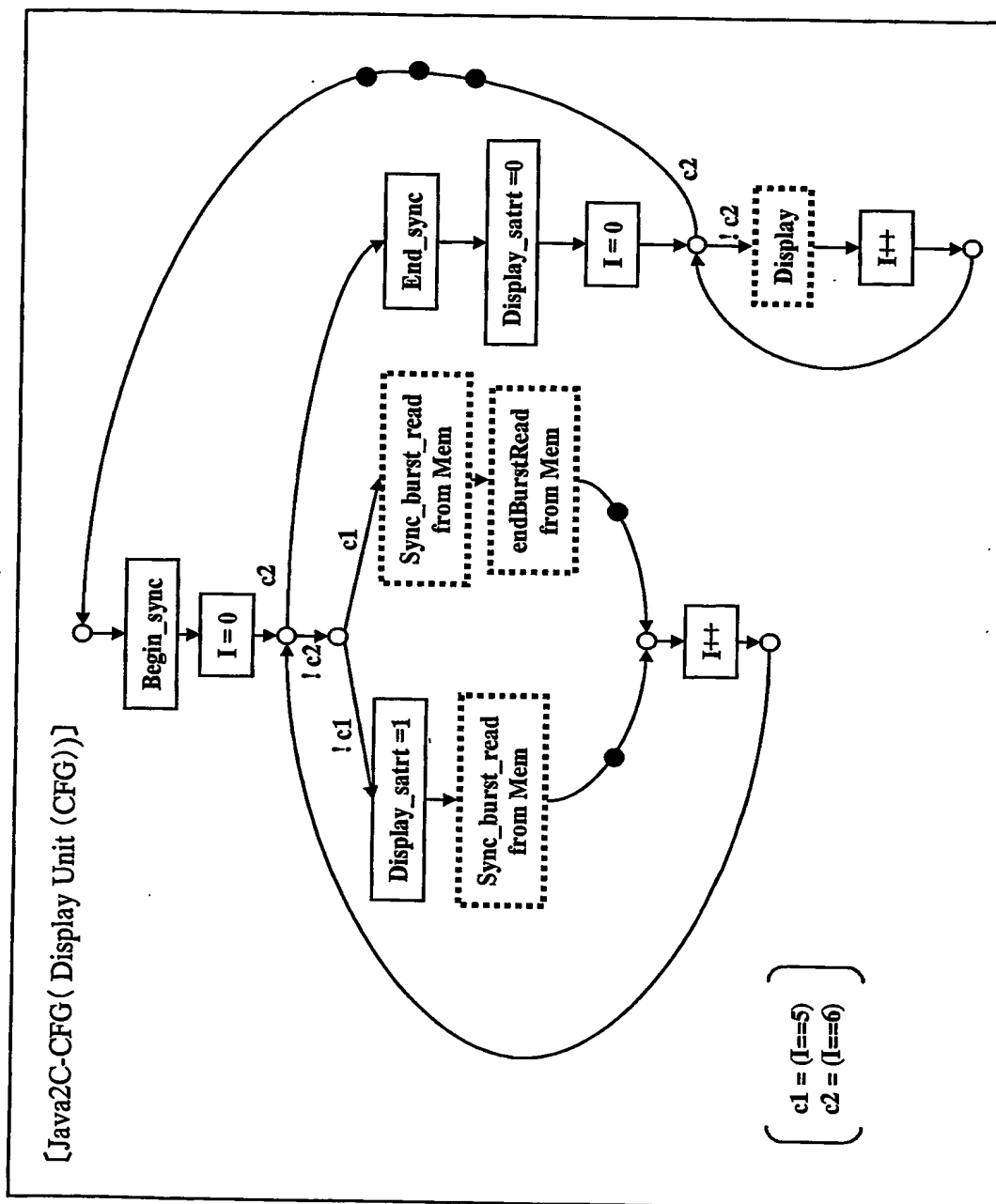
第45図





36 / 75

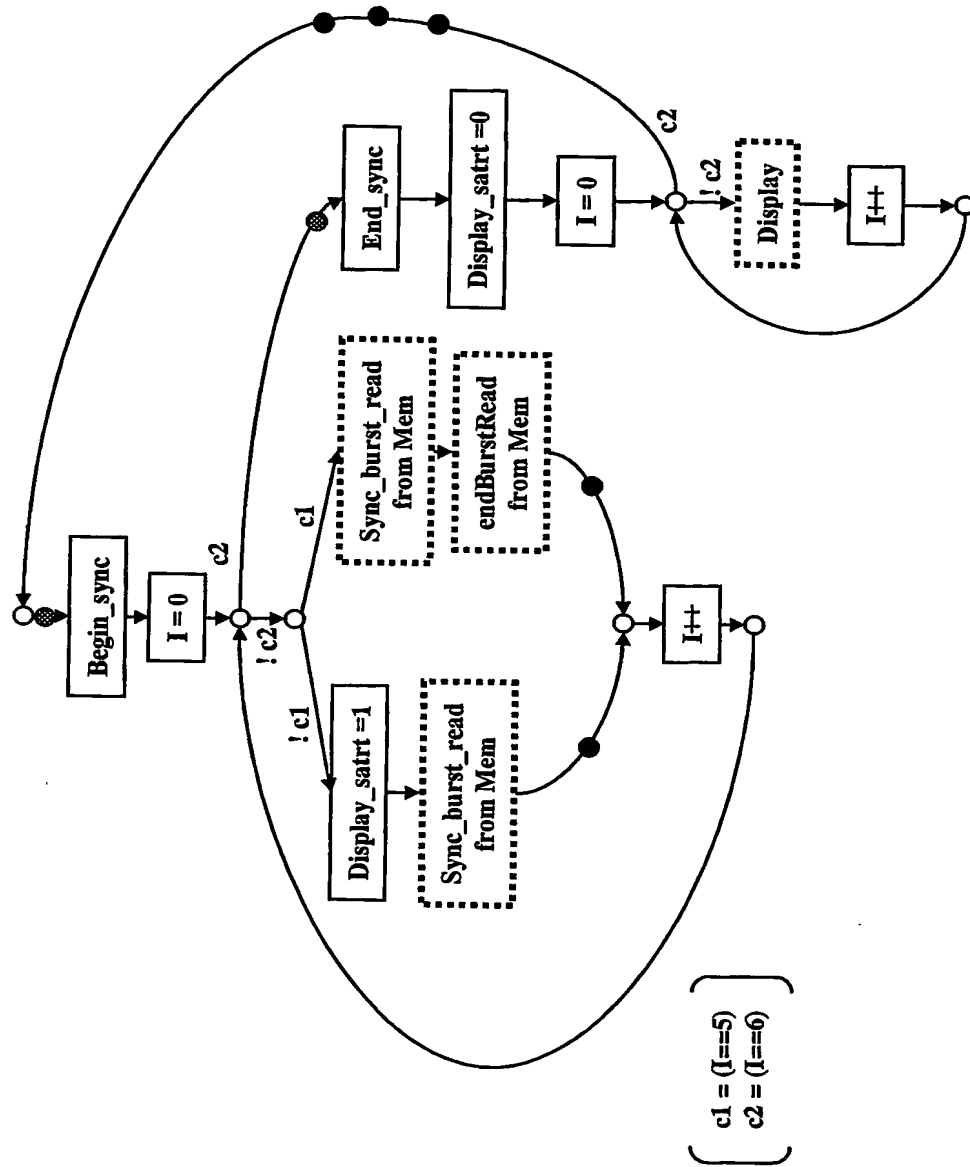
第46図



37/75

第47図

[Java2C-CFG (Display Unit (パラメトリック・モデルチェッキング用))]



38 / 75

第49図

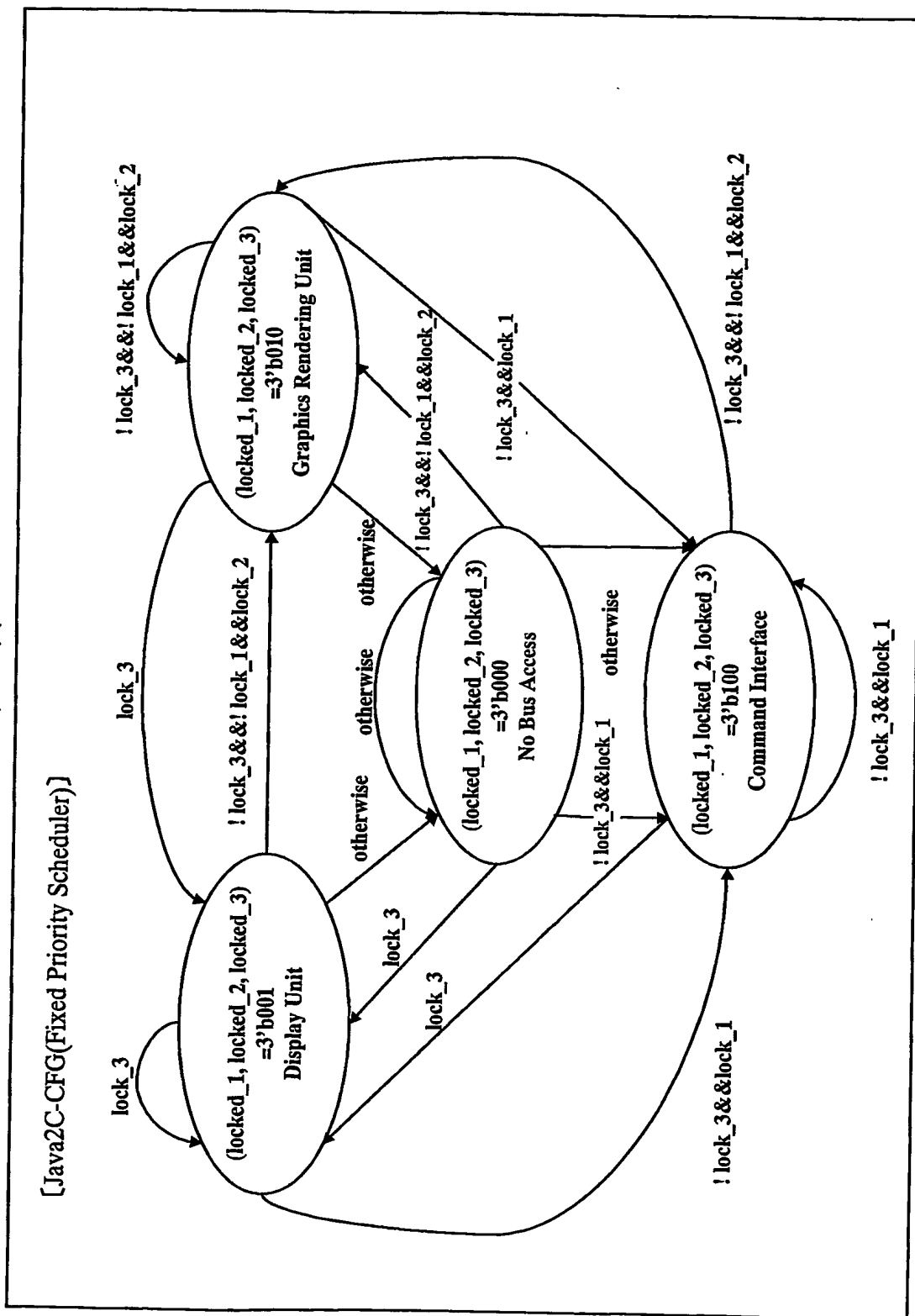
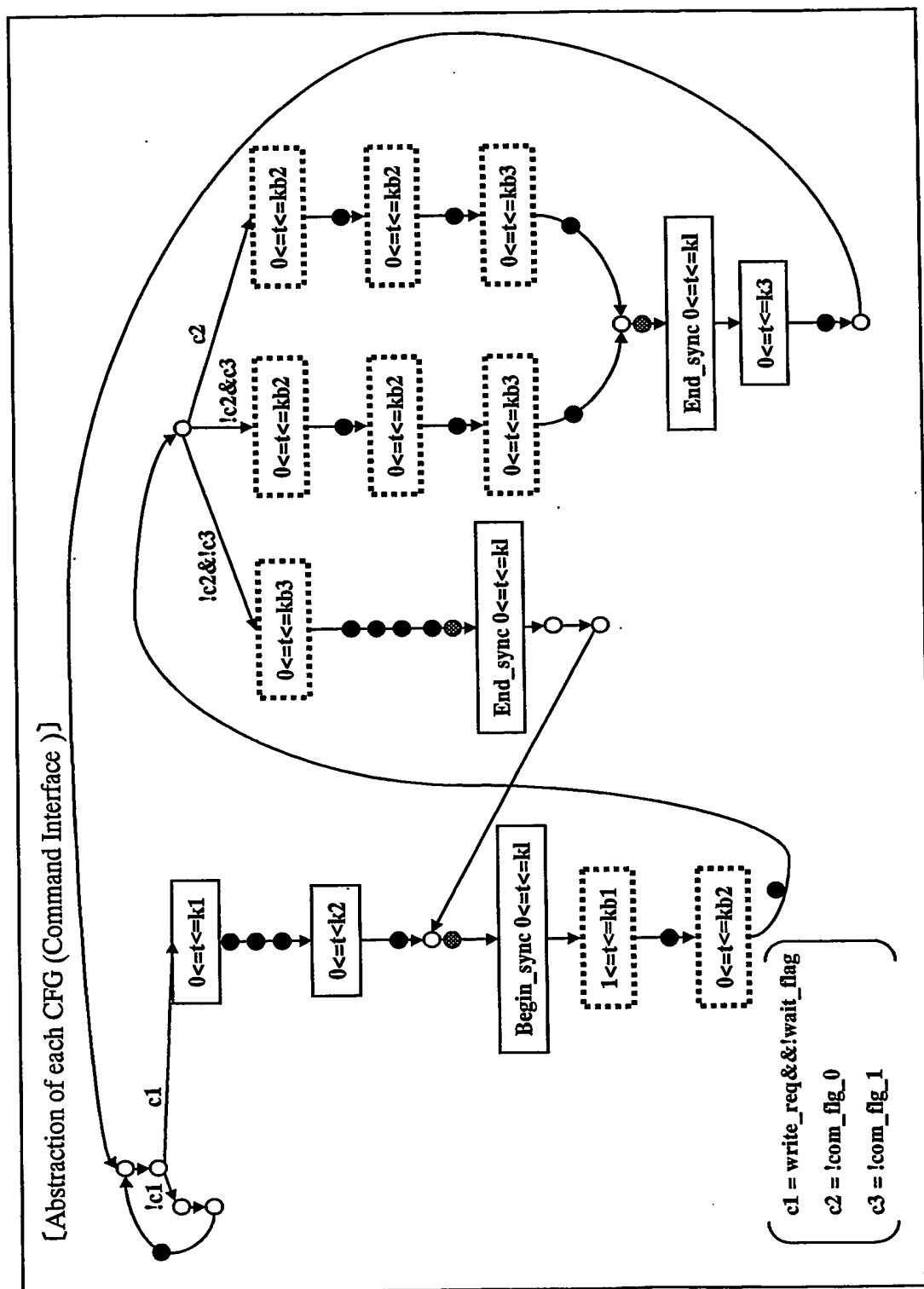
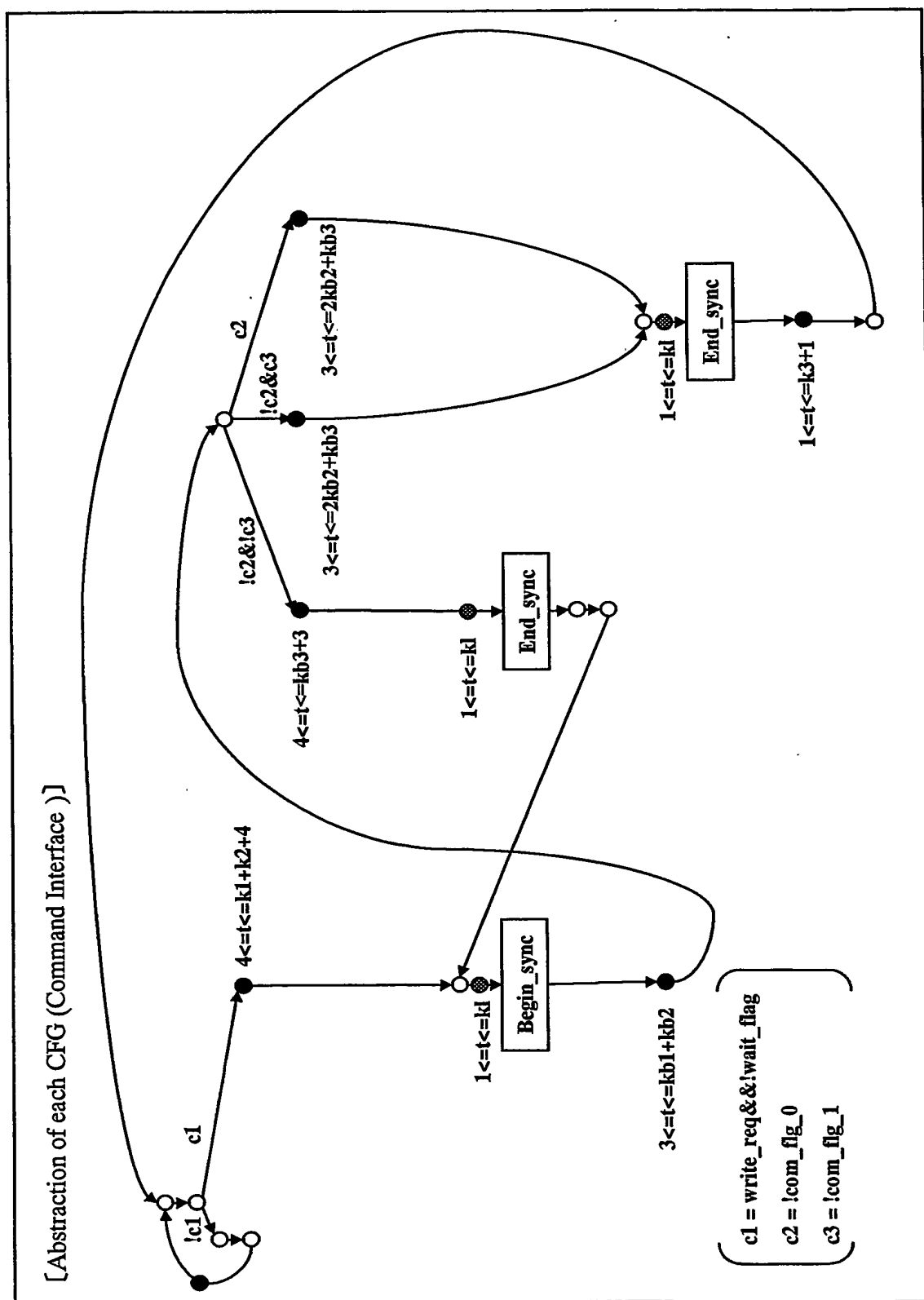


図 50 鋼



40/75

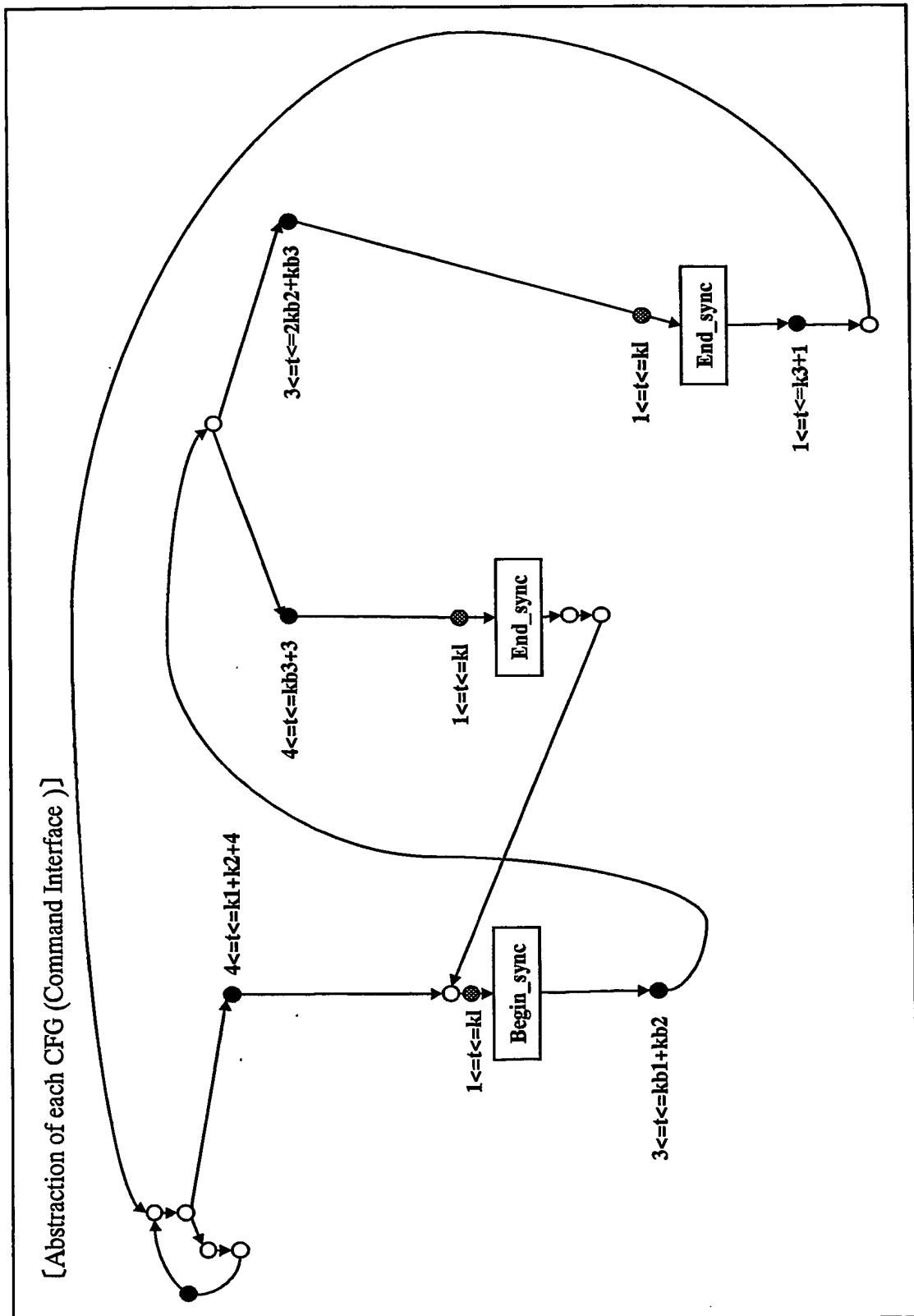
第 51 図



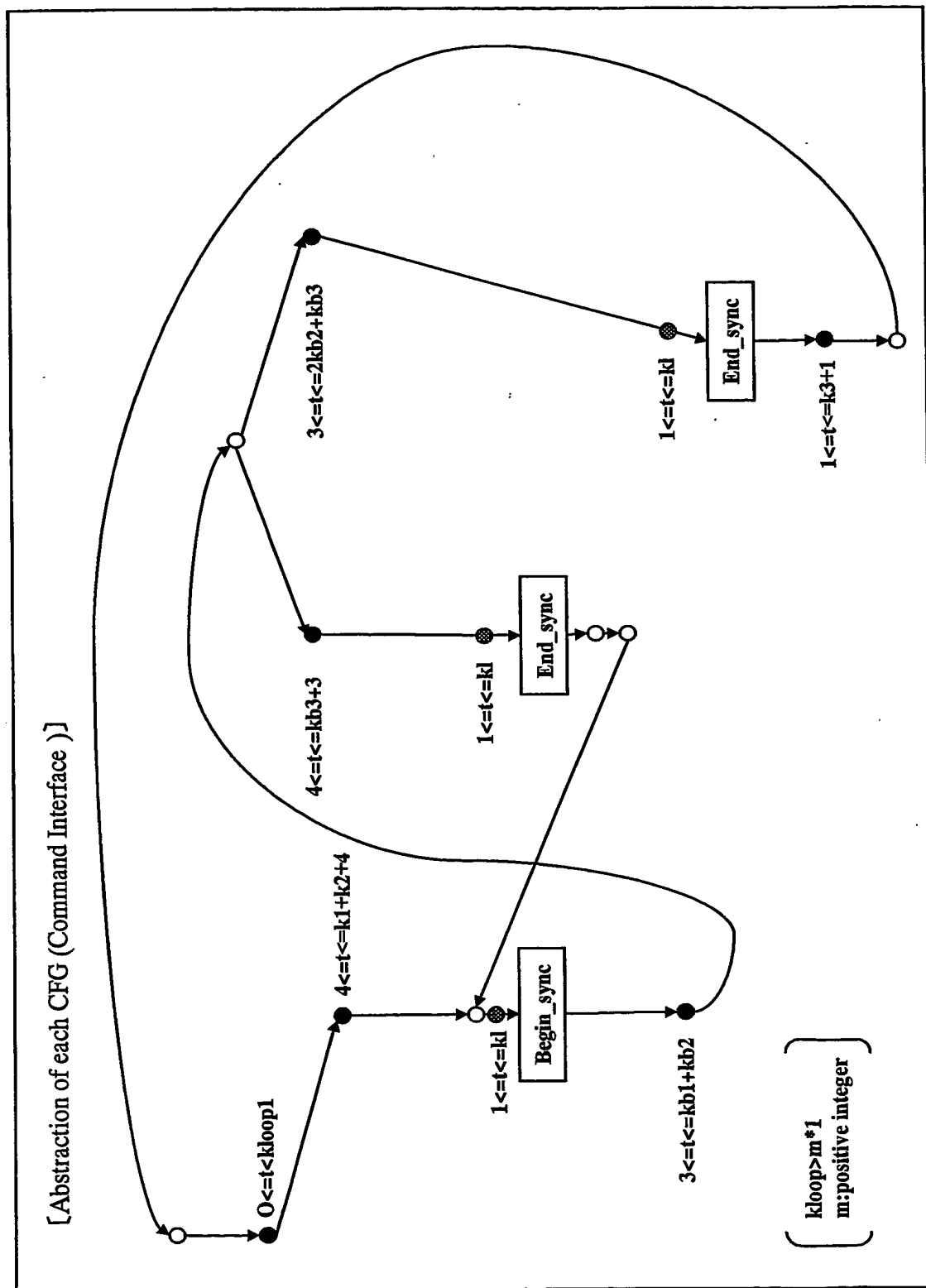
4 1 / 7 5

第 5 2 図

[Abstraction of each CFG (Command Interface )]

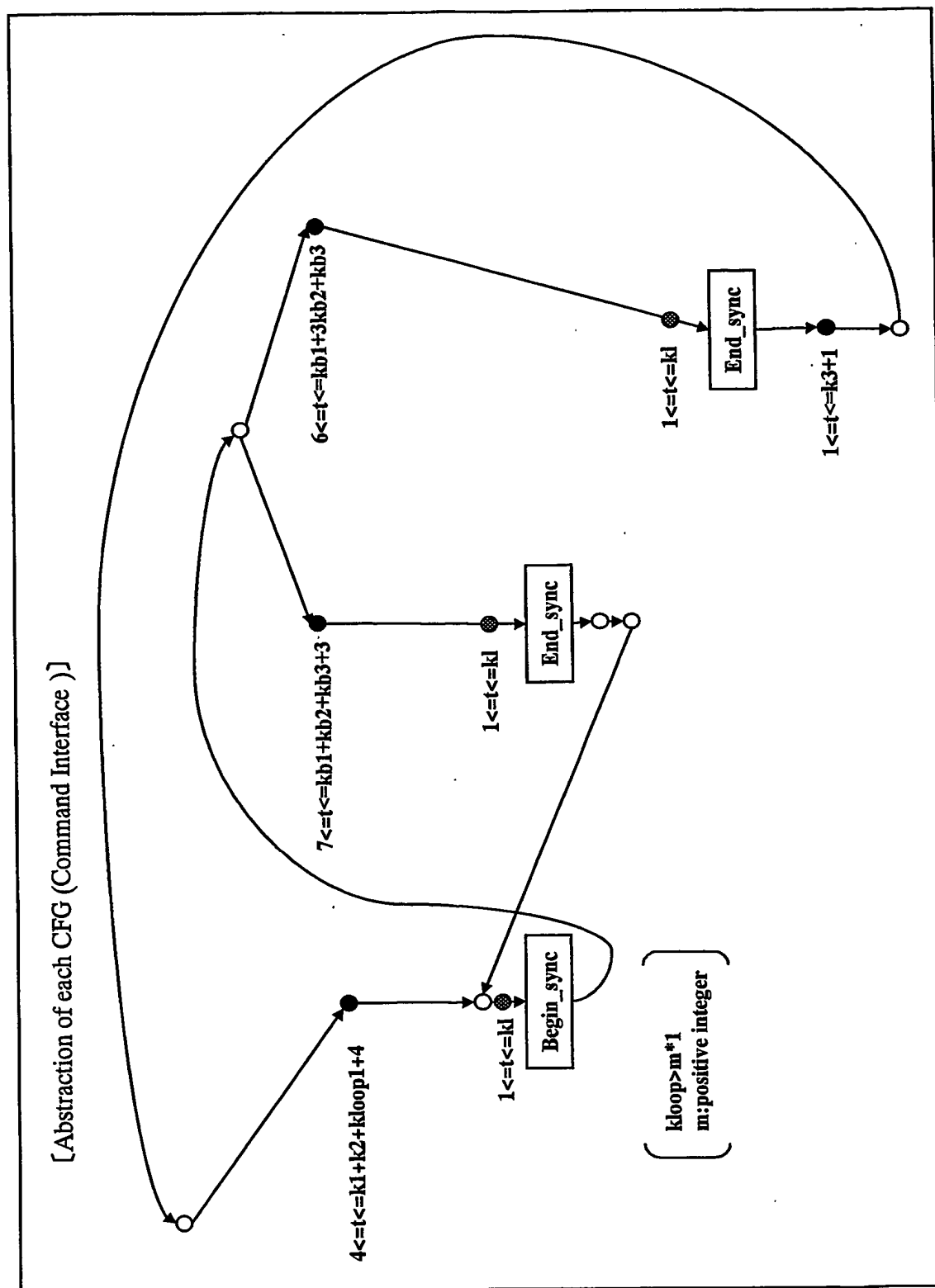


第三卷



43 / 75

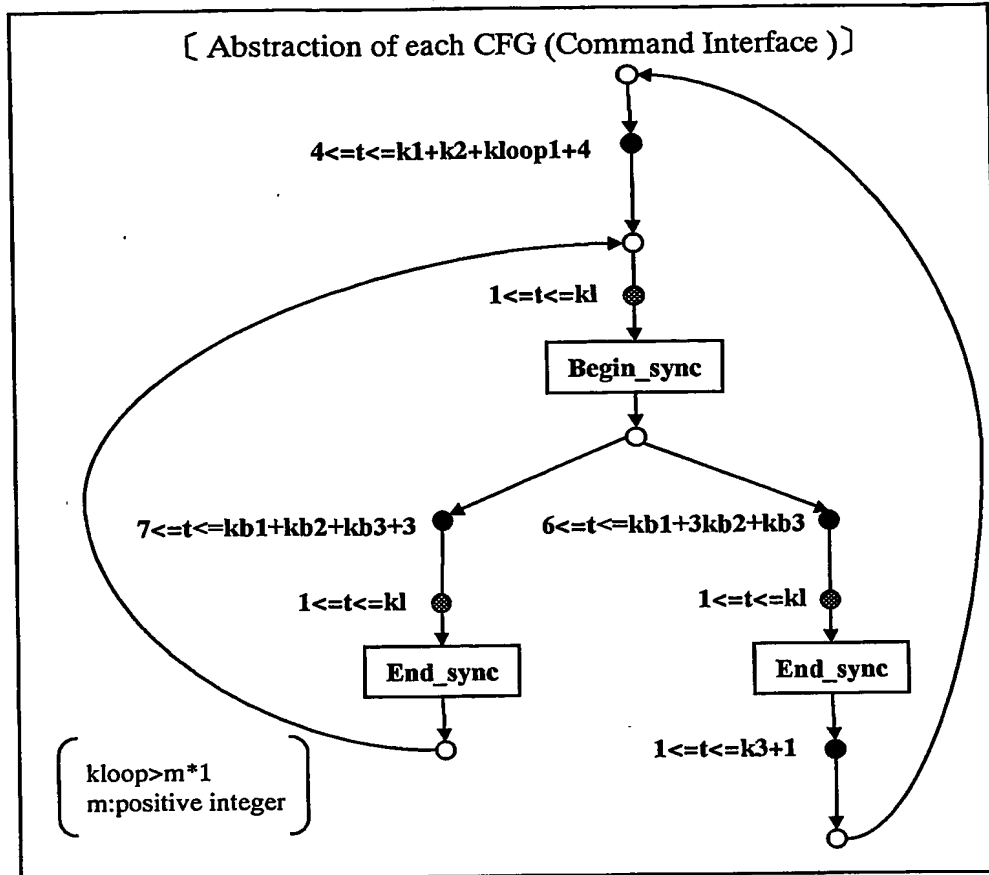
第 5 4 図



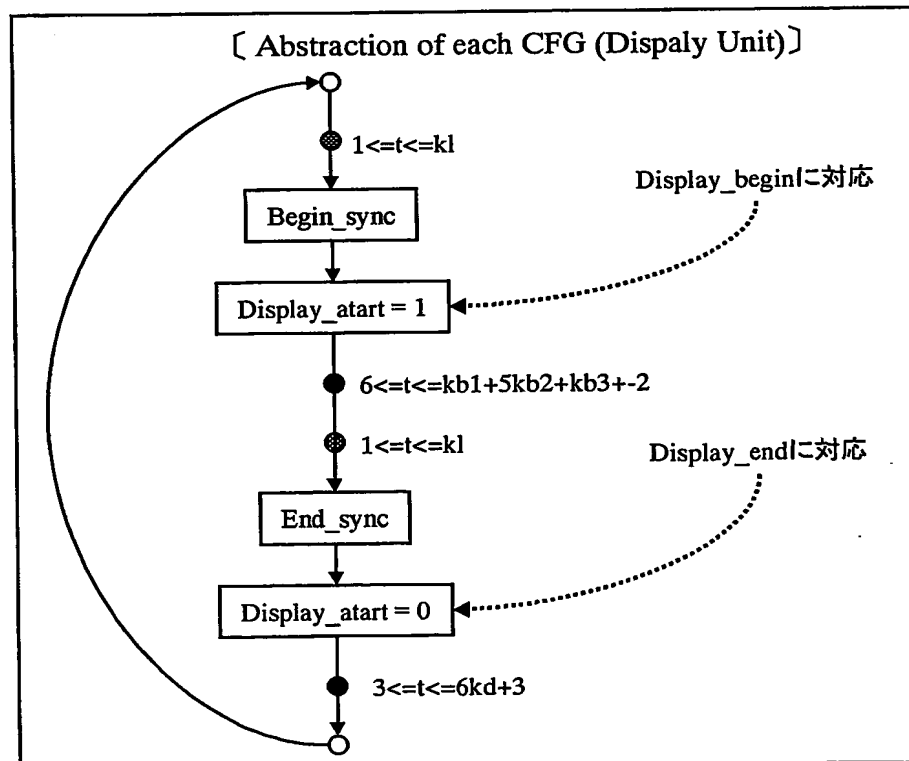


44 / 75

第55図

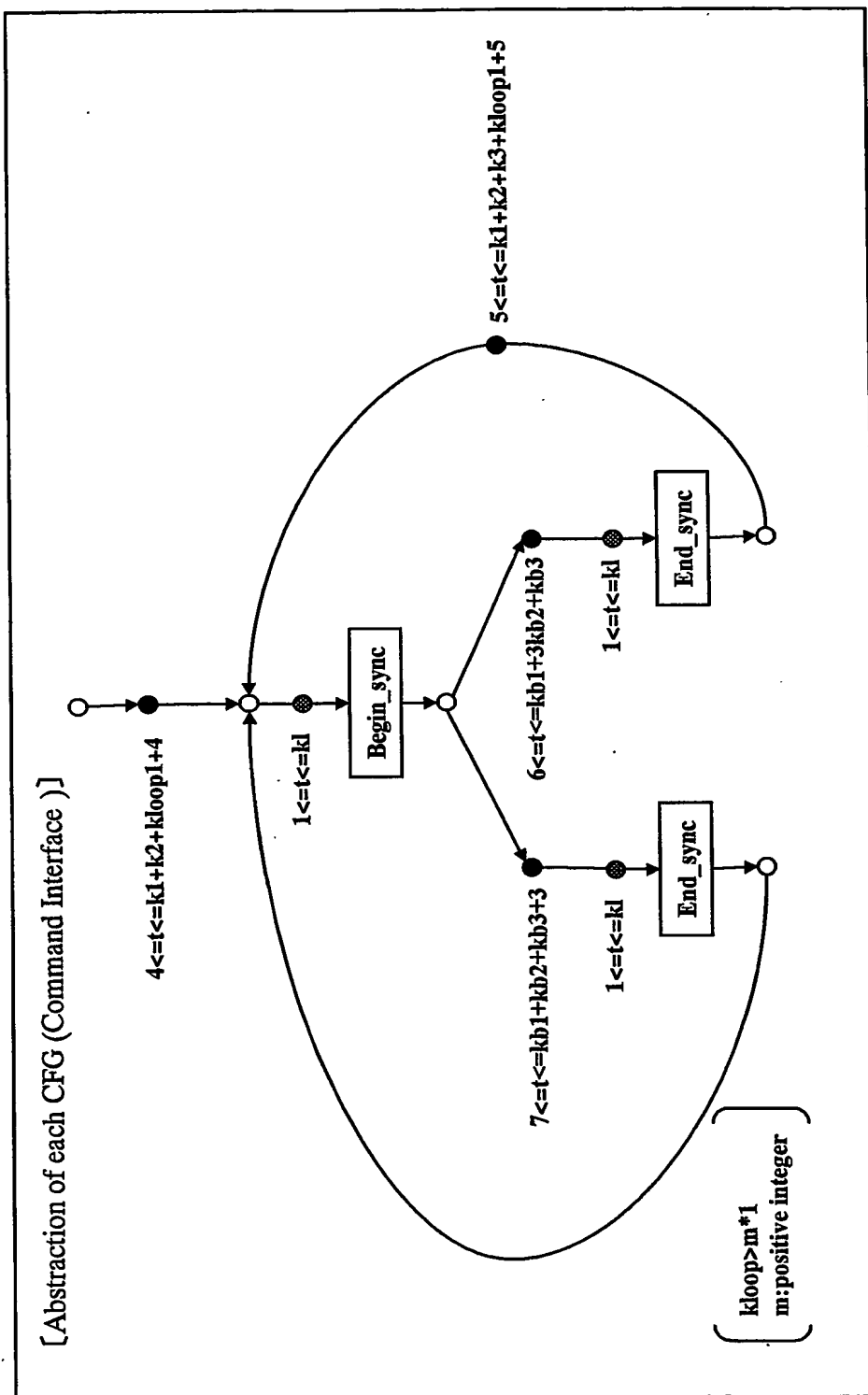


第58図



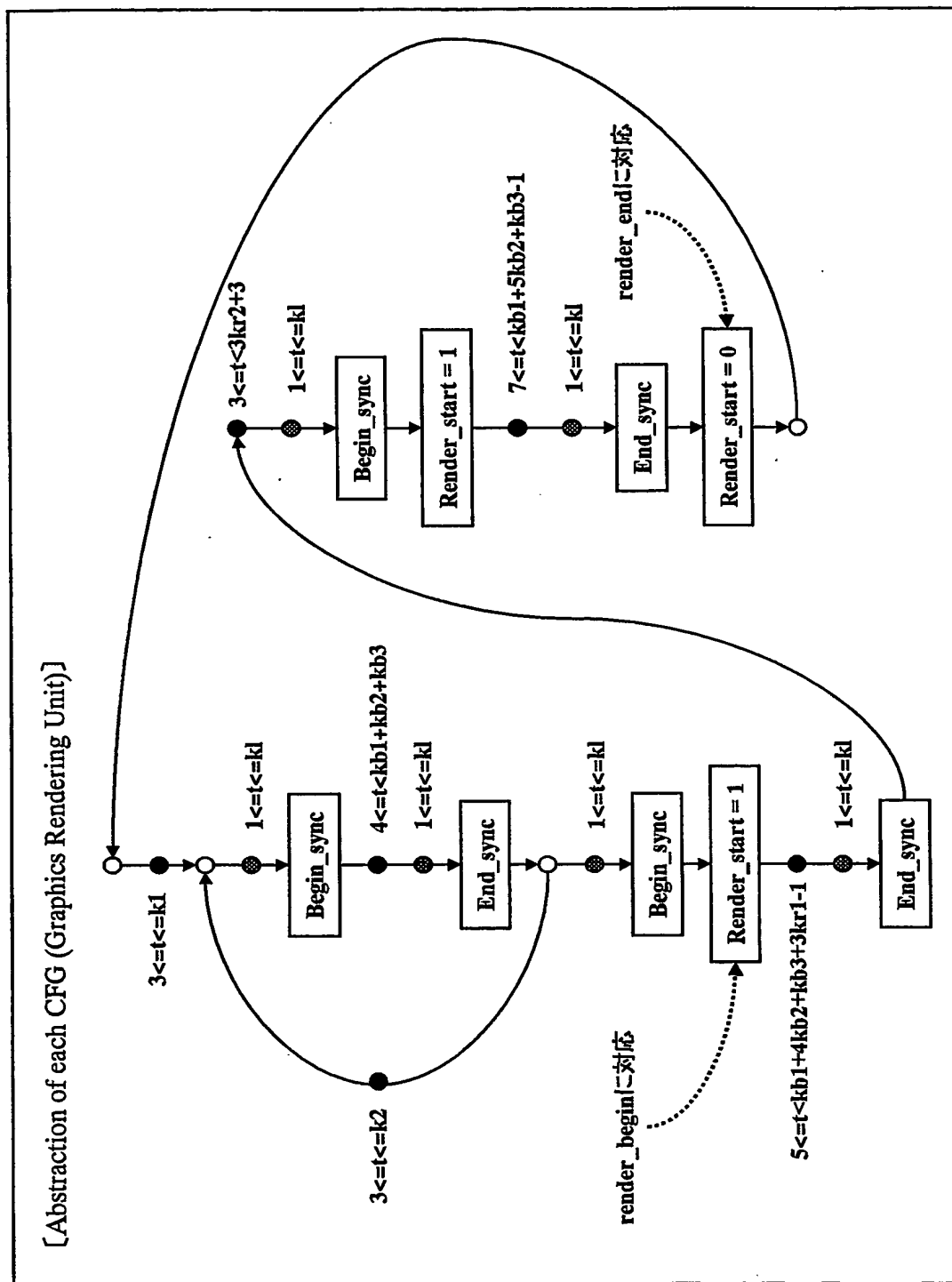
45 / 75

第 5 6 図



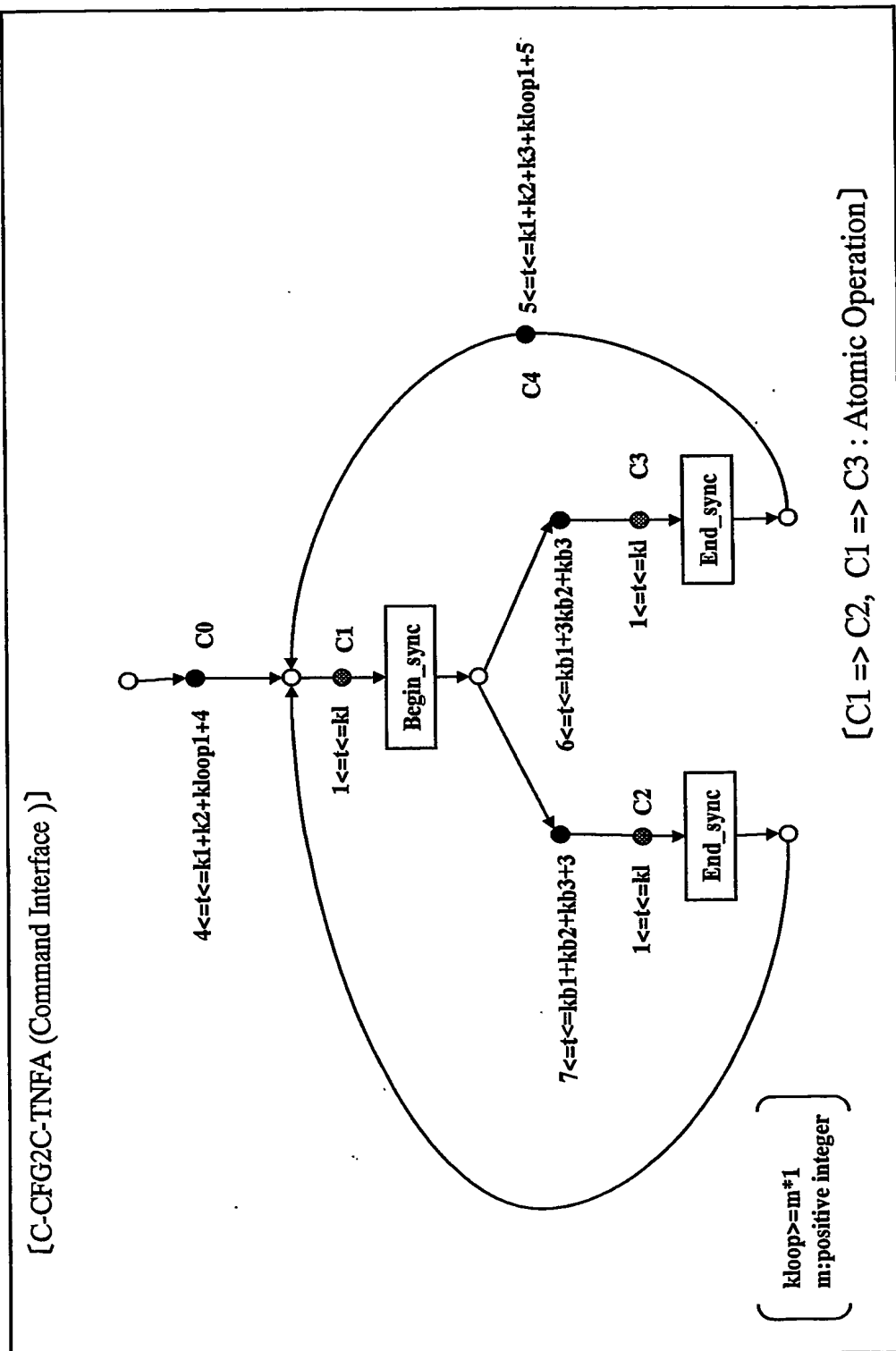
46 / 75

第57図



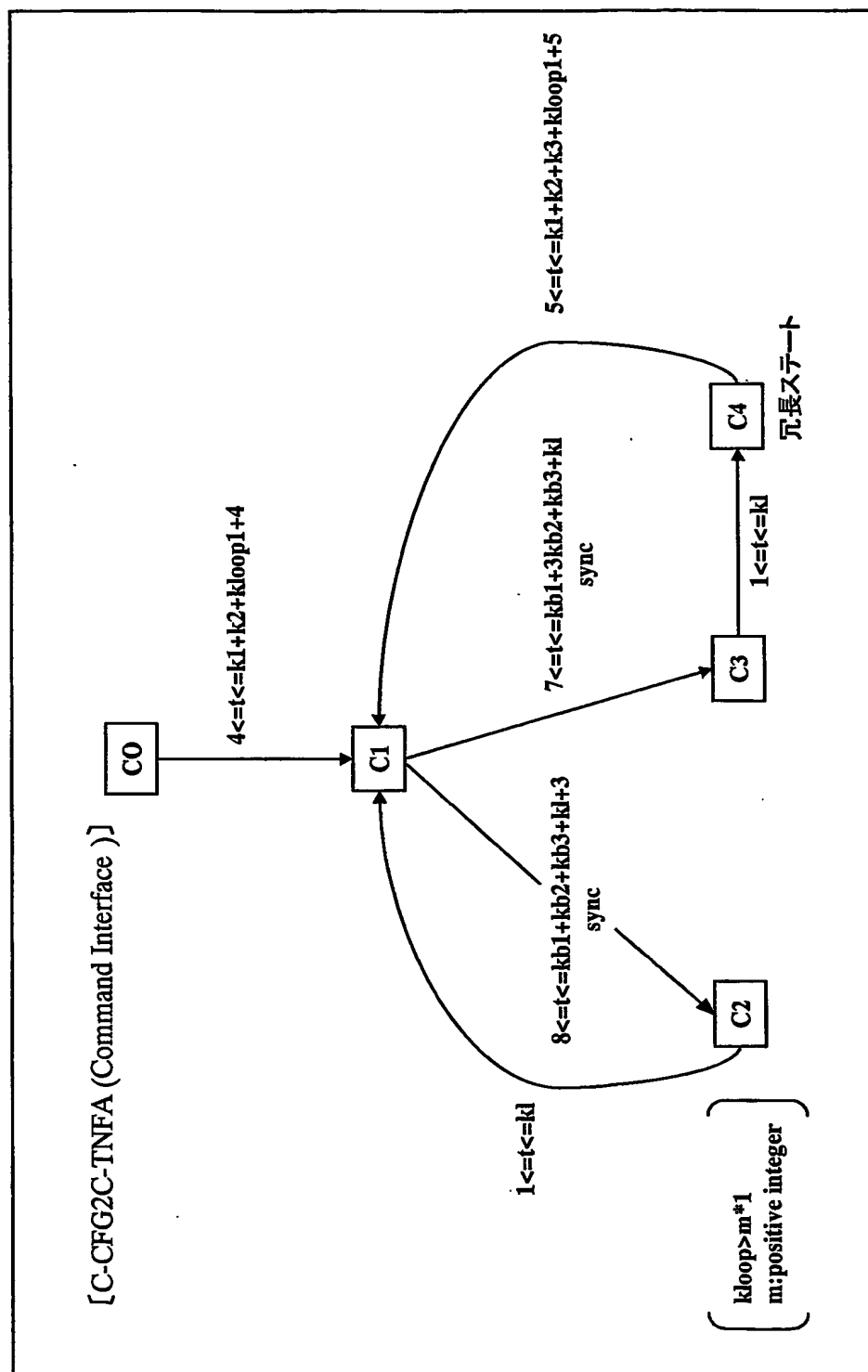
47/75

第59図

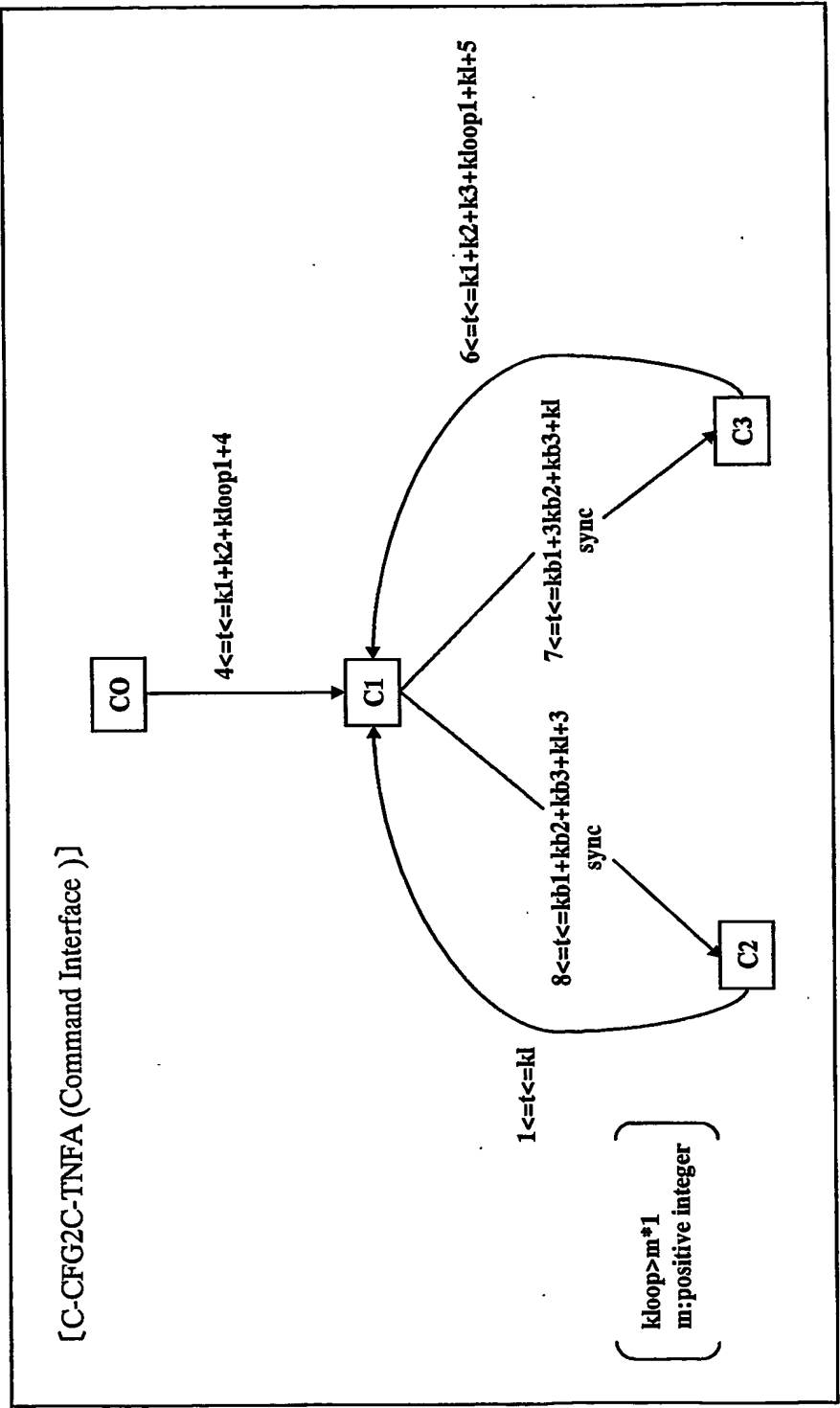


48 / 75

第60図

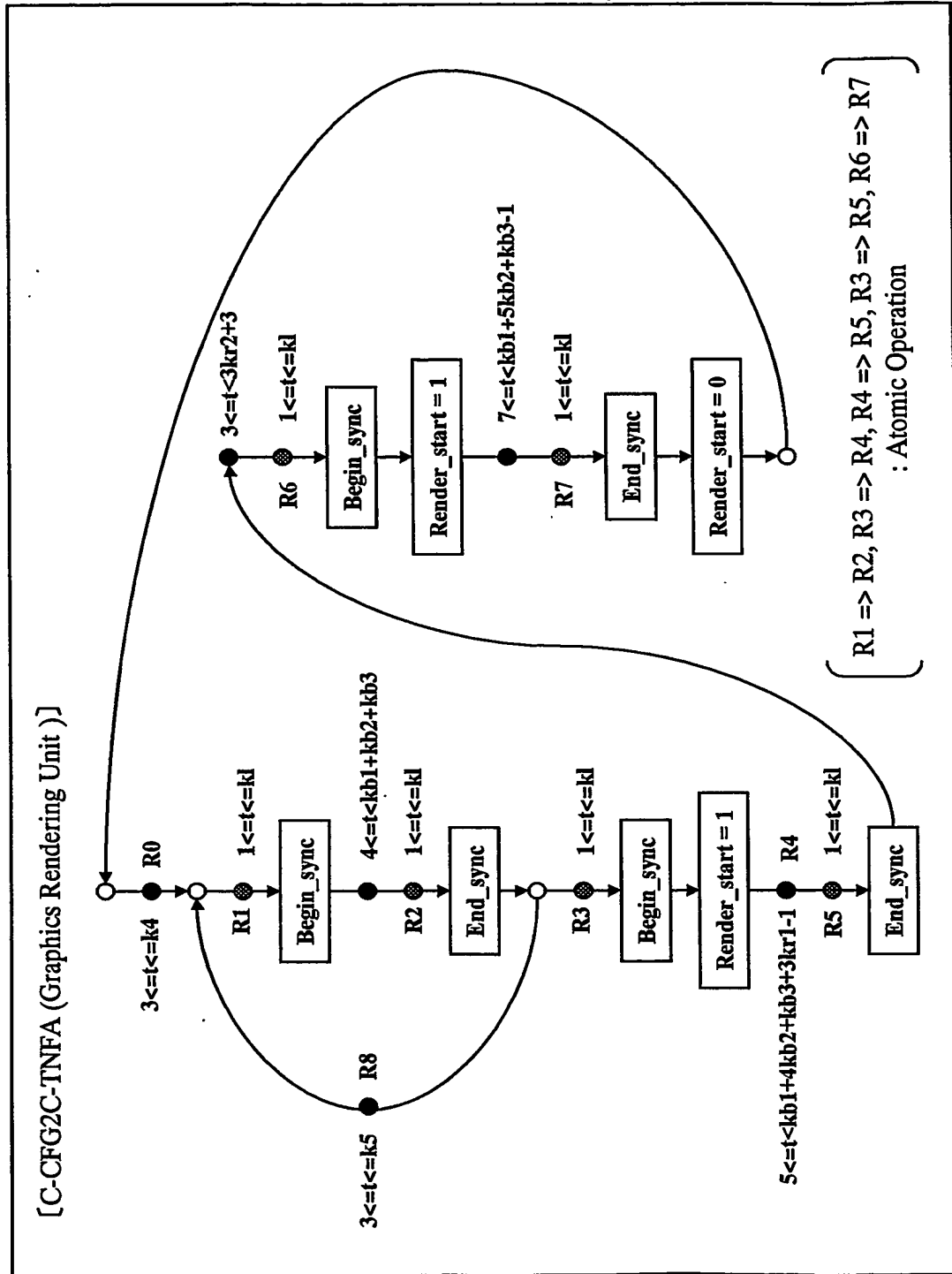


第 61 図



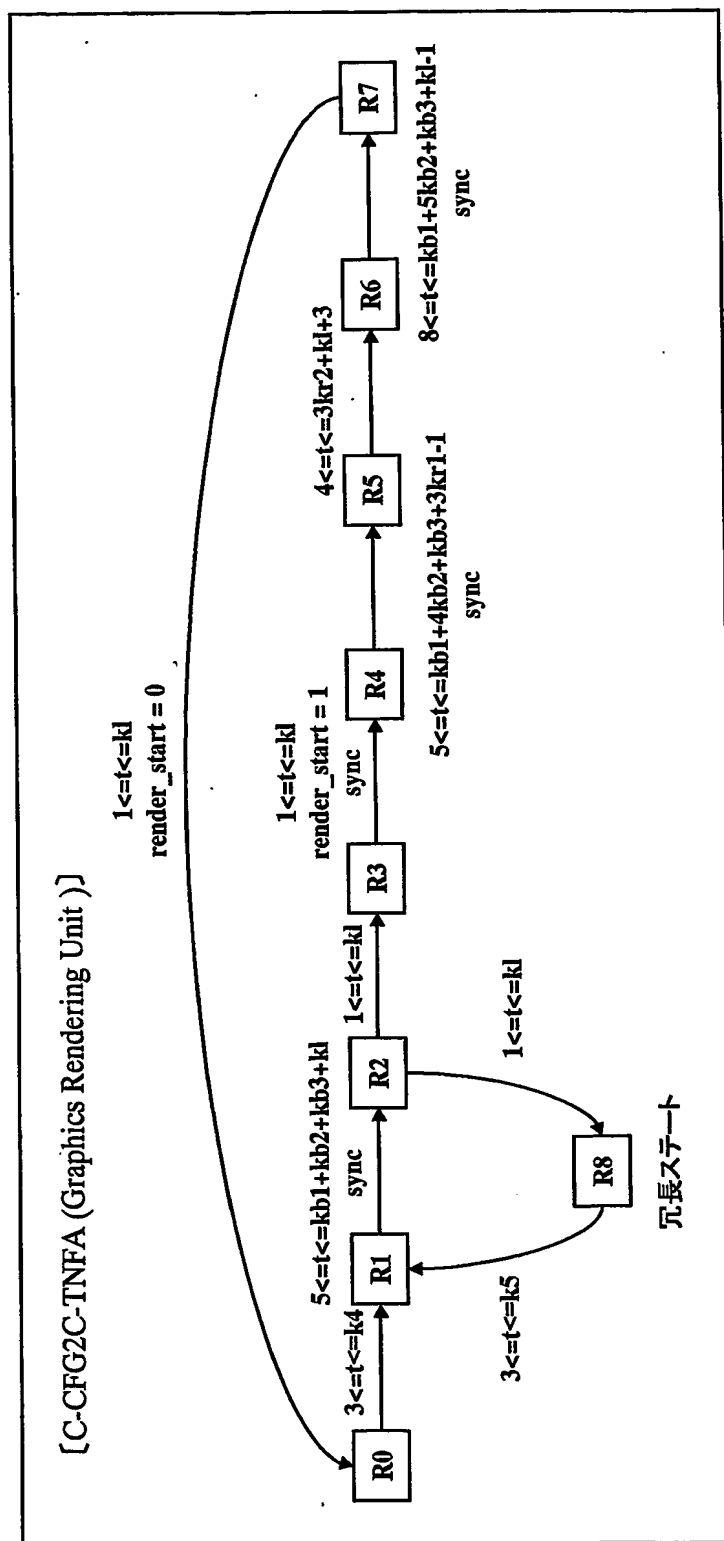
50/75

第 6 2 図



51 / 75

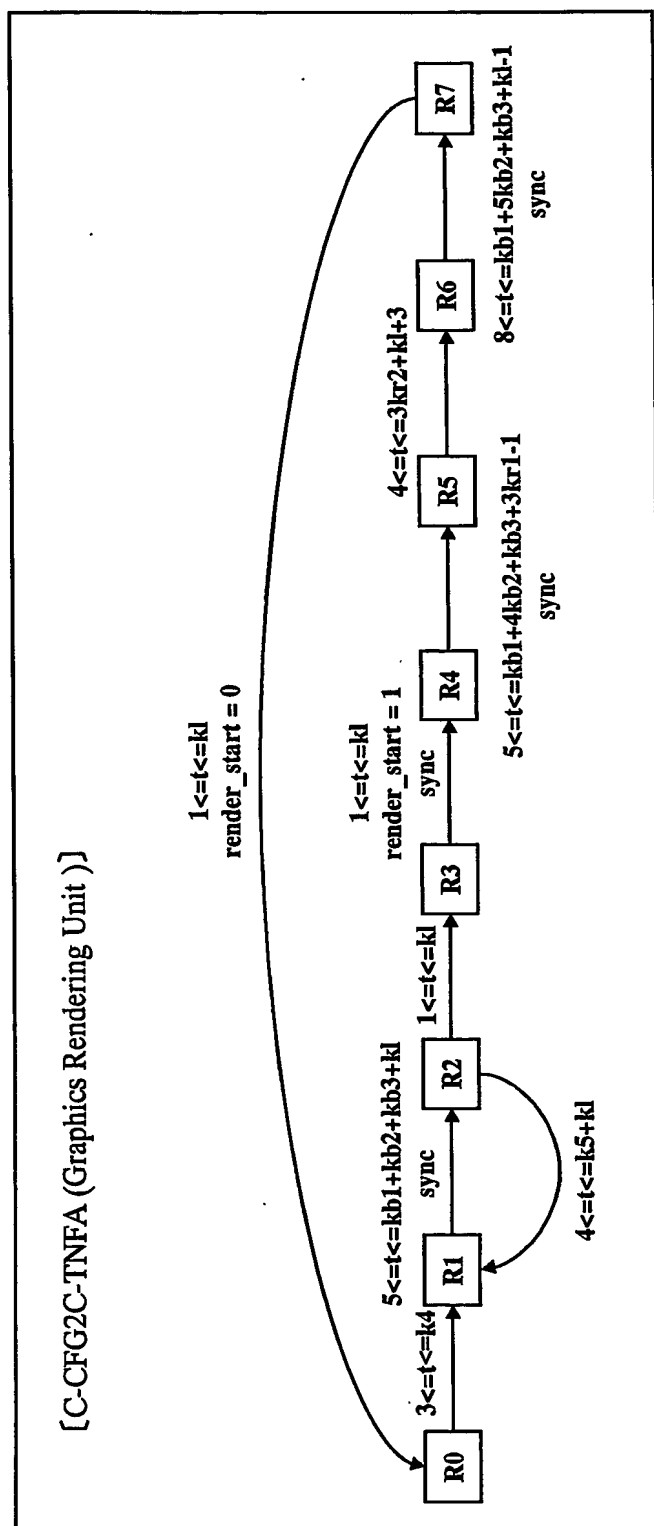
第 6 3 図



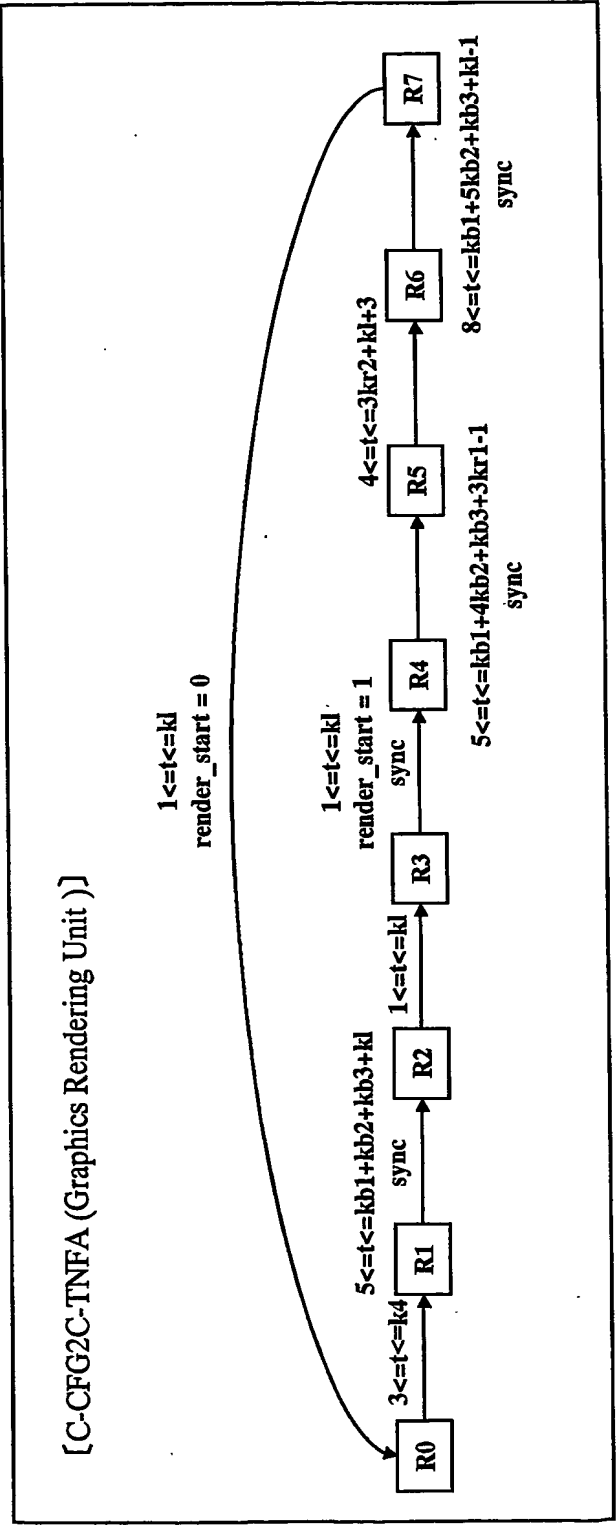


52 / 75

第 6 4 図

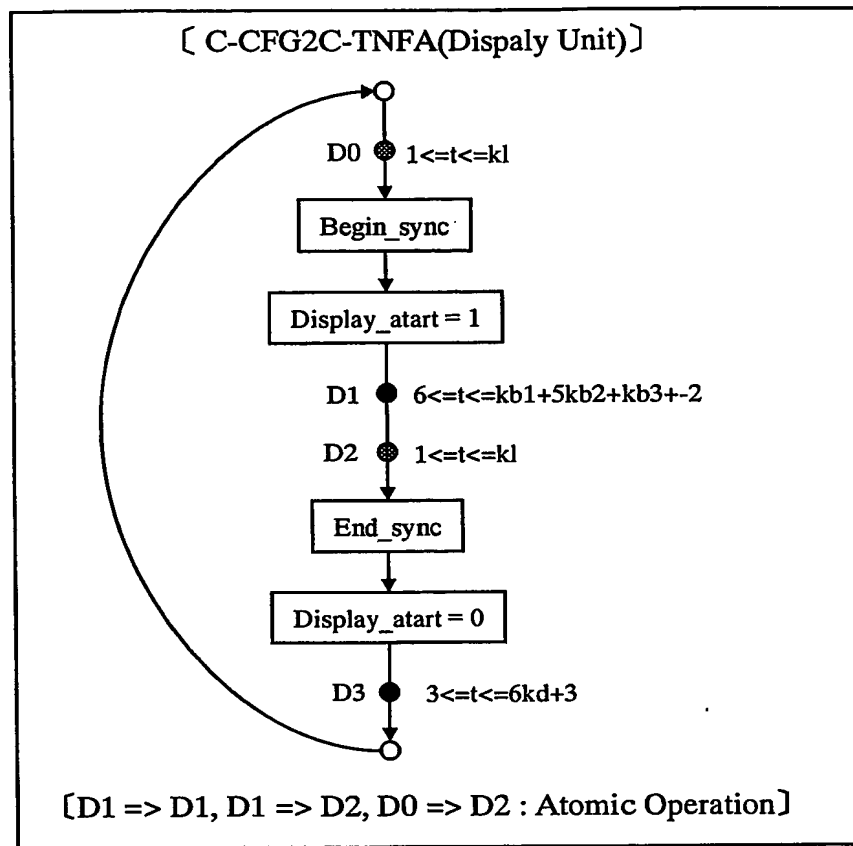


第 6 5 図

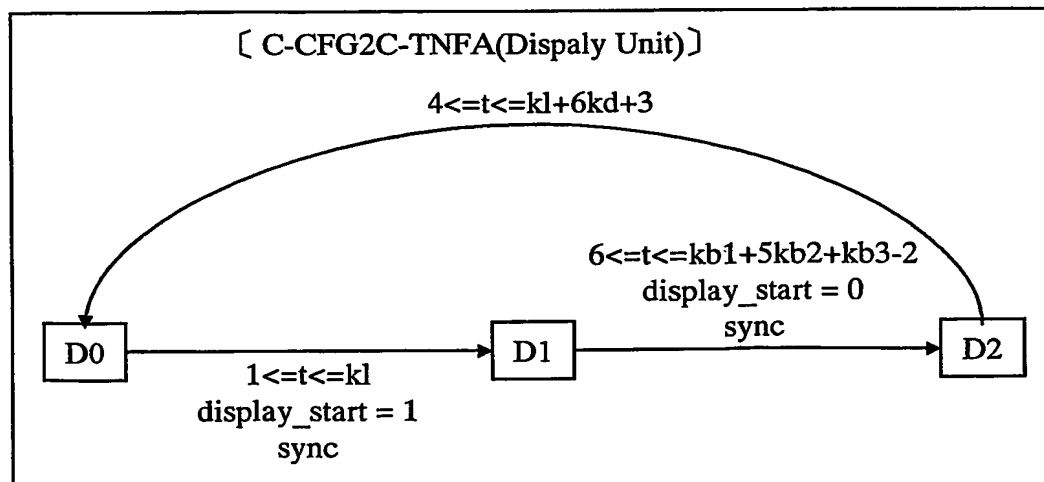


5 4 / 7 5

第 6 6 図

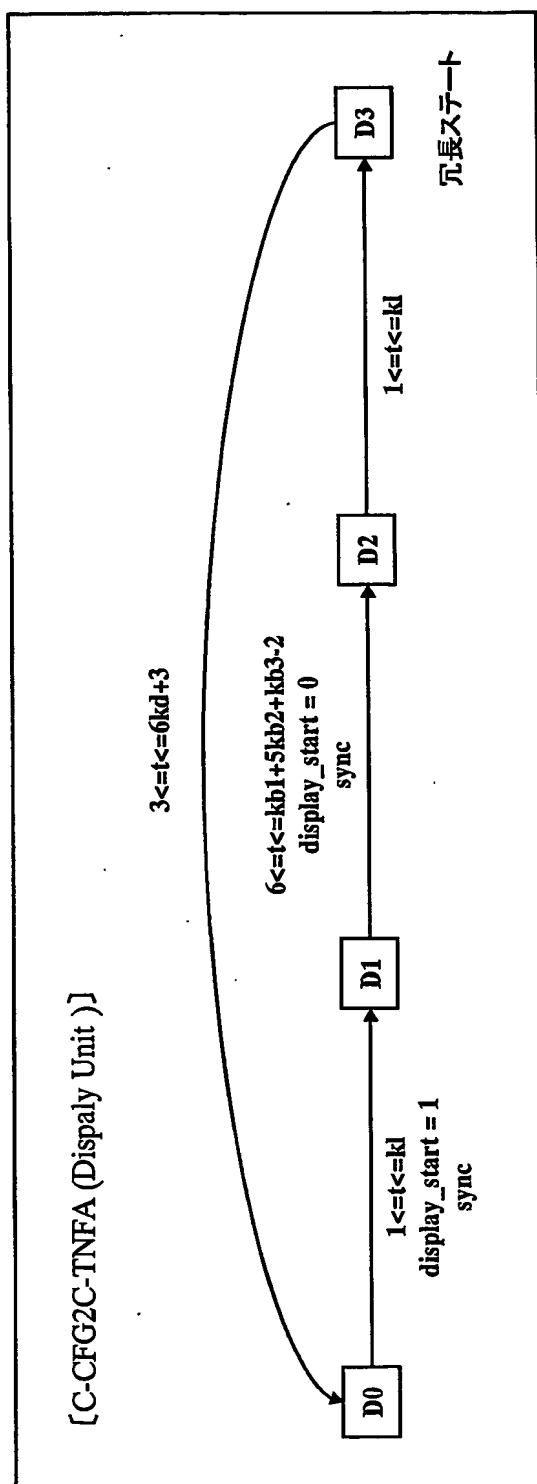


第 6 8 図

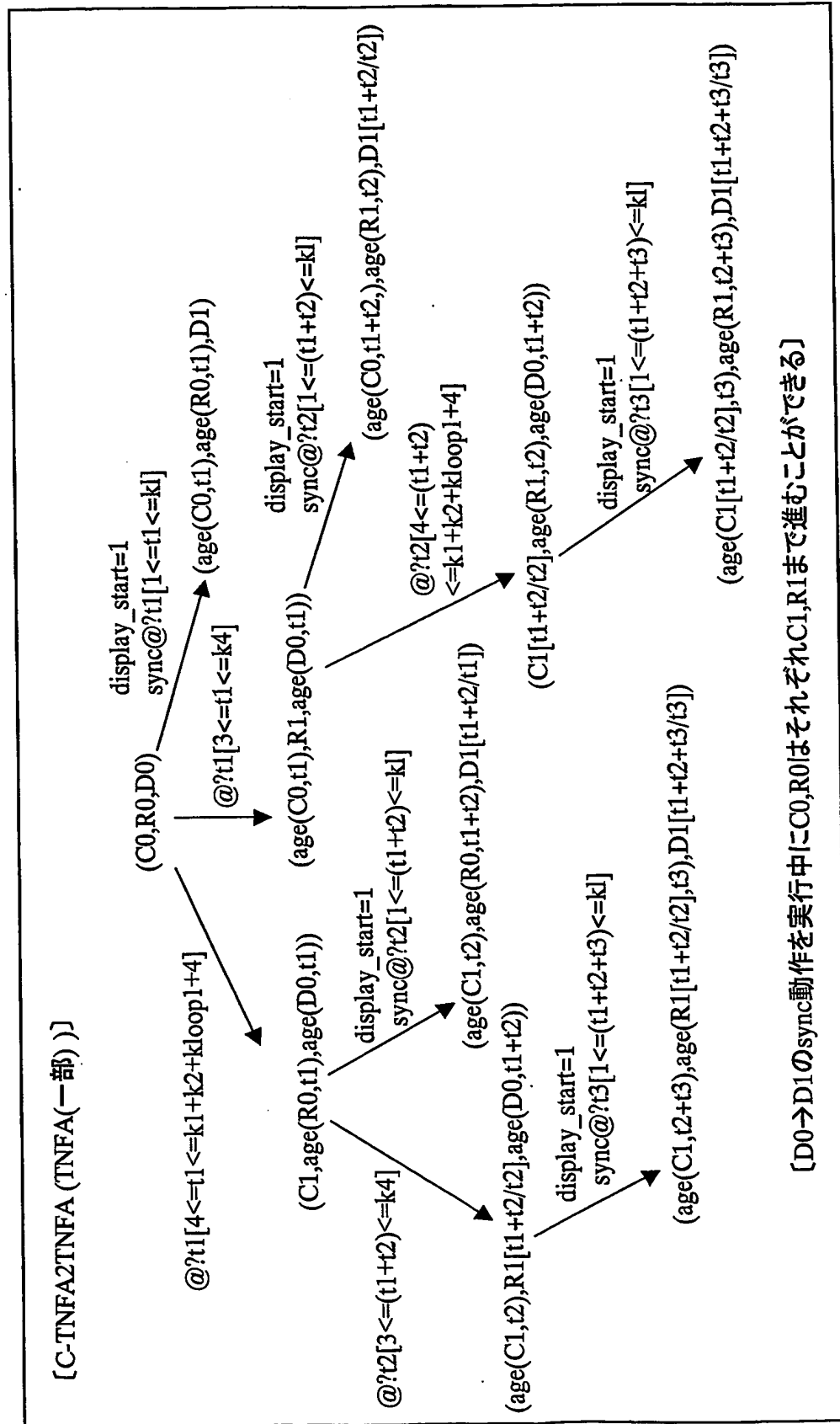


55 / 75

第 67 図



第69図

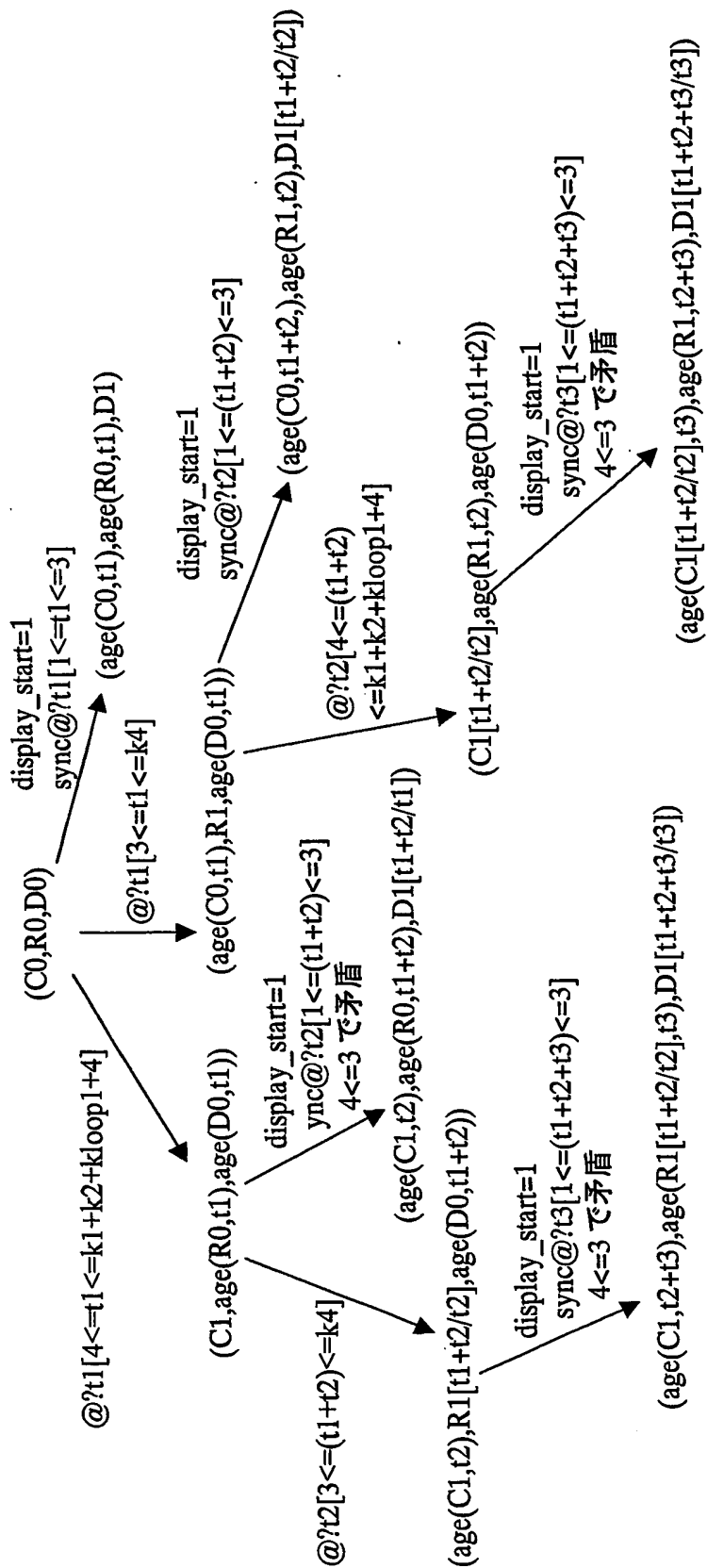




58 / 75

第71図

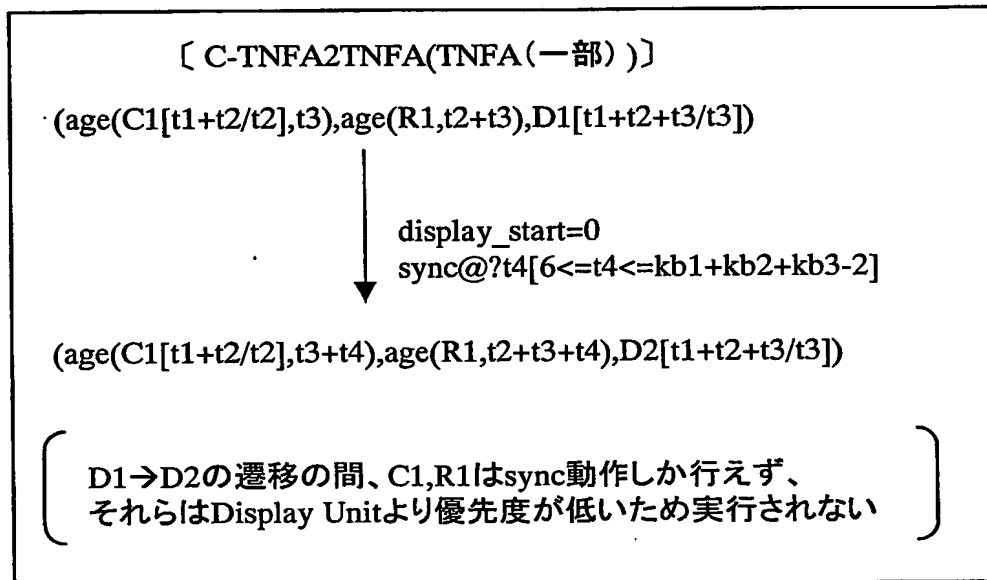
[C-TNFA2TNFA (TNFA(一部))]



[時間制約を満たさない遷移枝へのみ到達する状態を削除する]

59 / 75

## 第72図



## 第80図

〔パラメトリック解析結果(実行結果例)〕

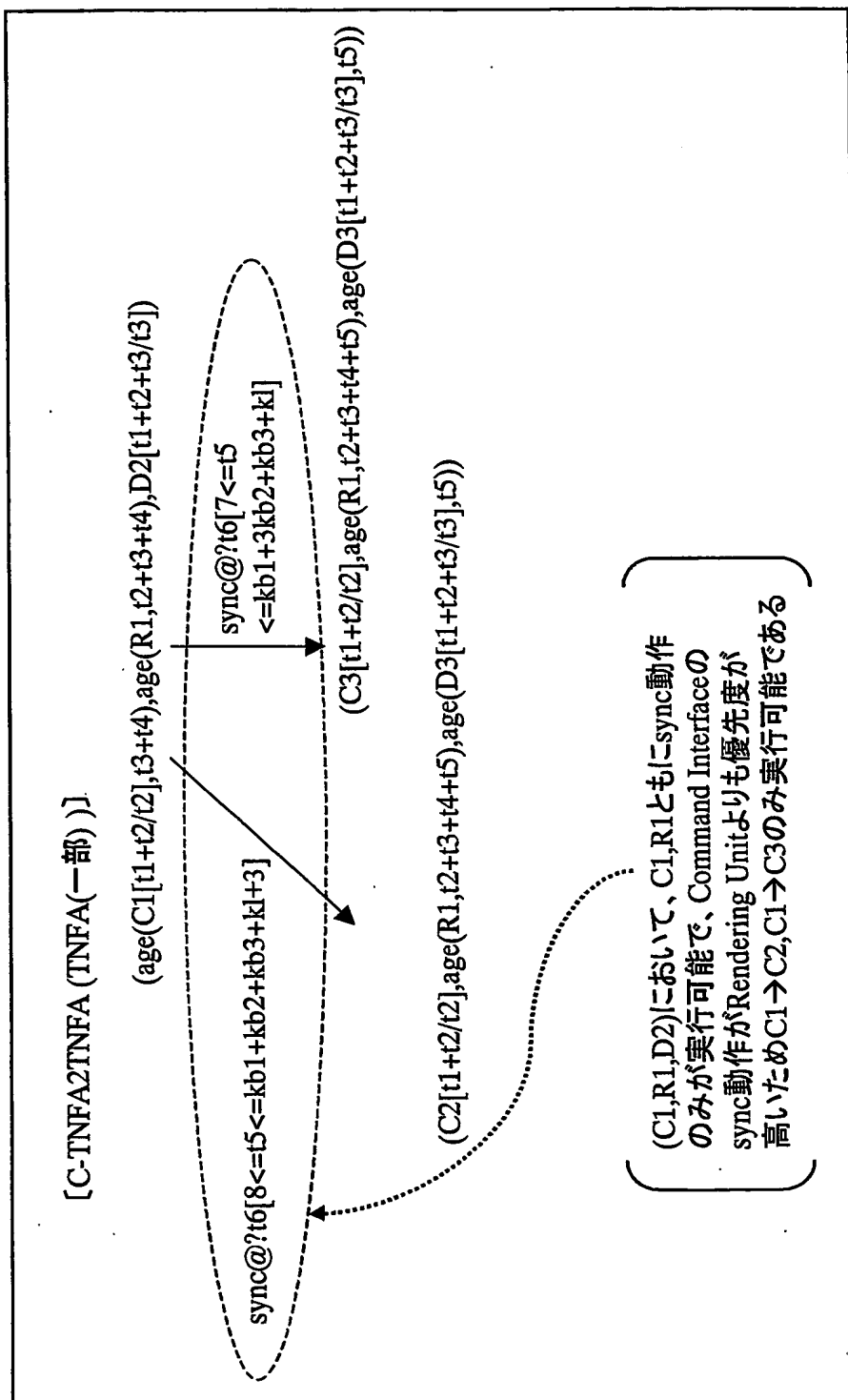
Value of objective function: 12

k1	0
k2	0
k3	0
k4	4
k5	0
kd	0
kl	4
kloop1	0
kb1	3
kb2	1
kb3	0
kr1	0
kr2	0

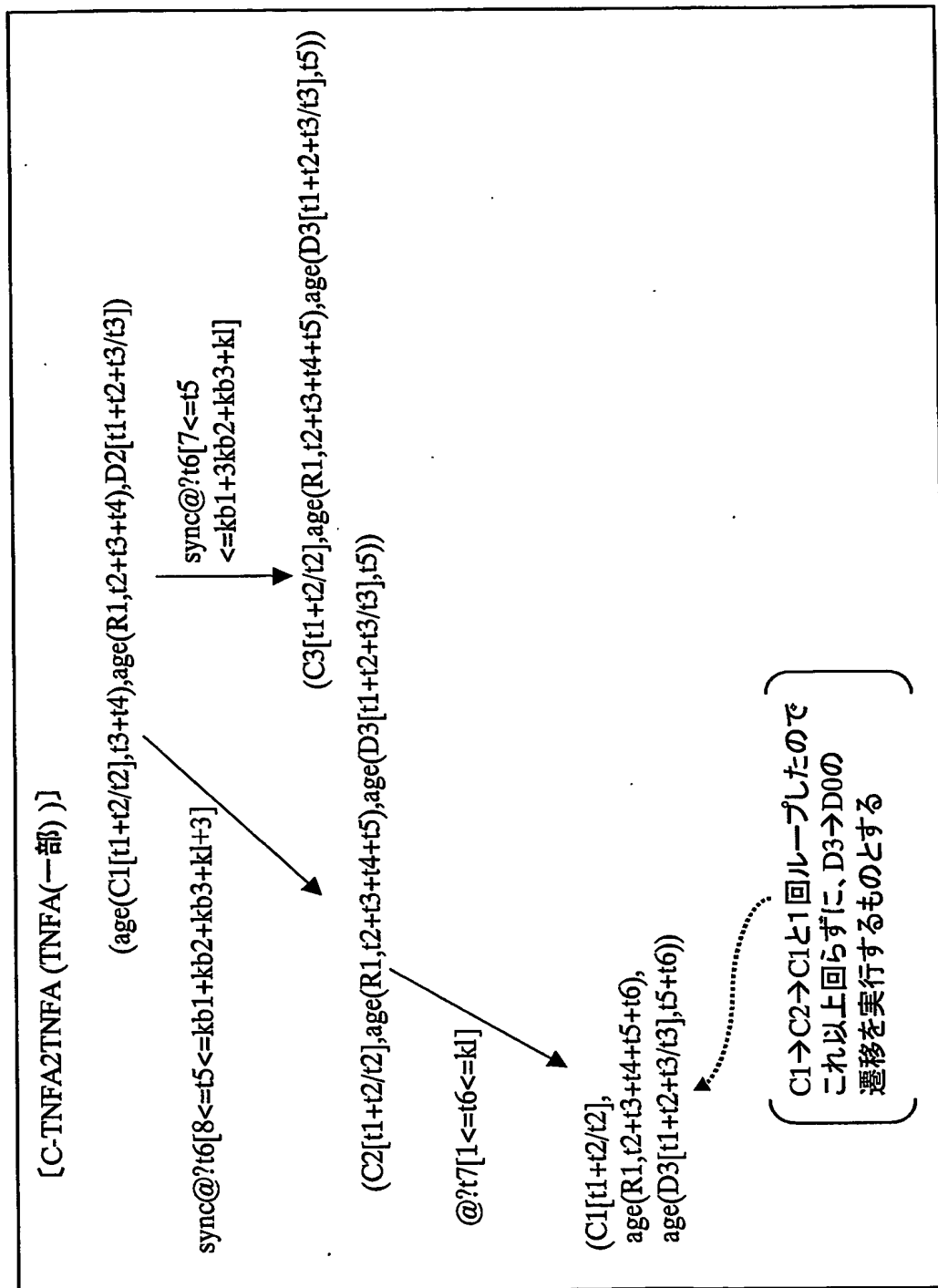


60/75

第73図



第74図



第75圖

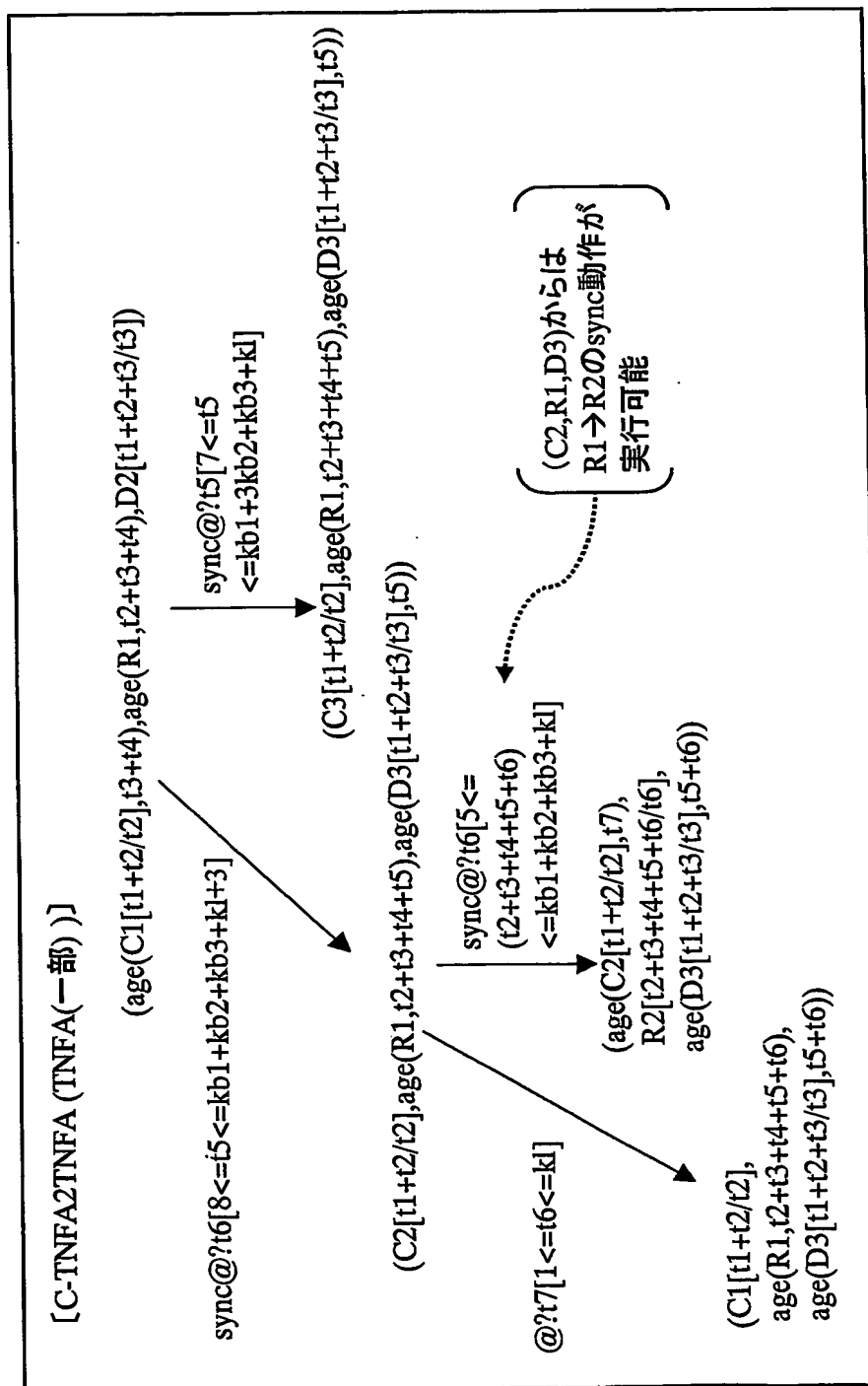
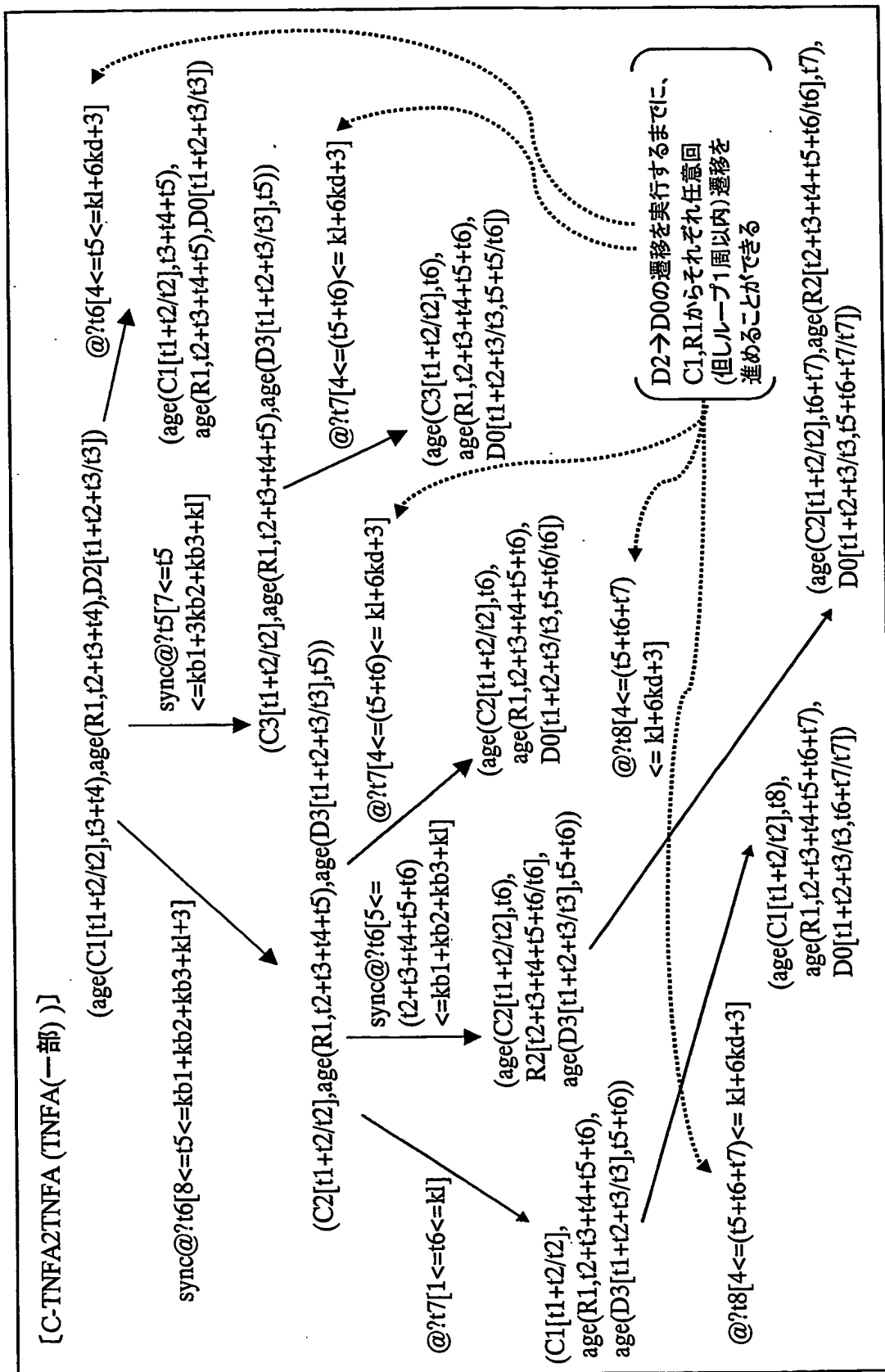
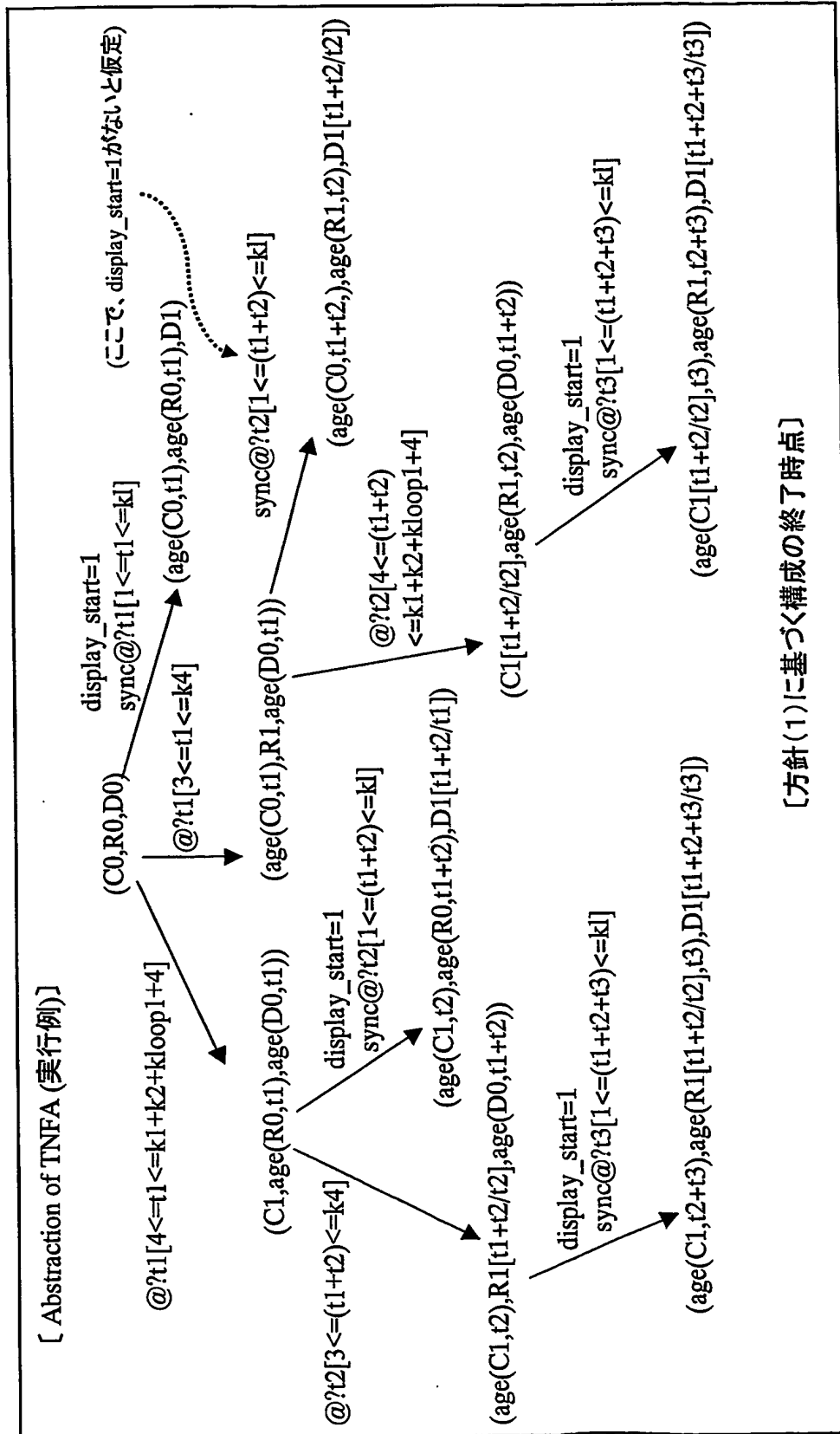


図 67 振

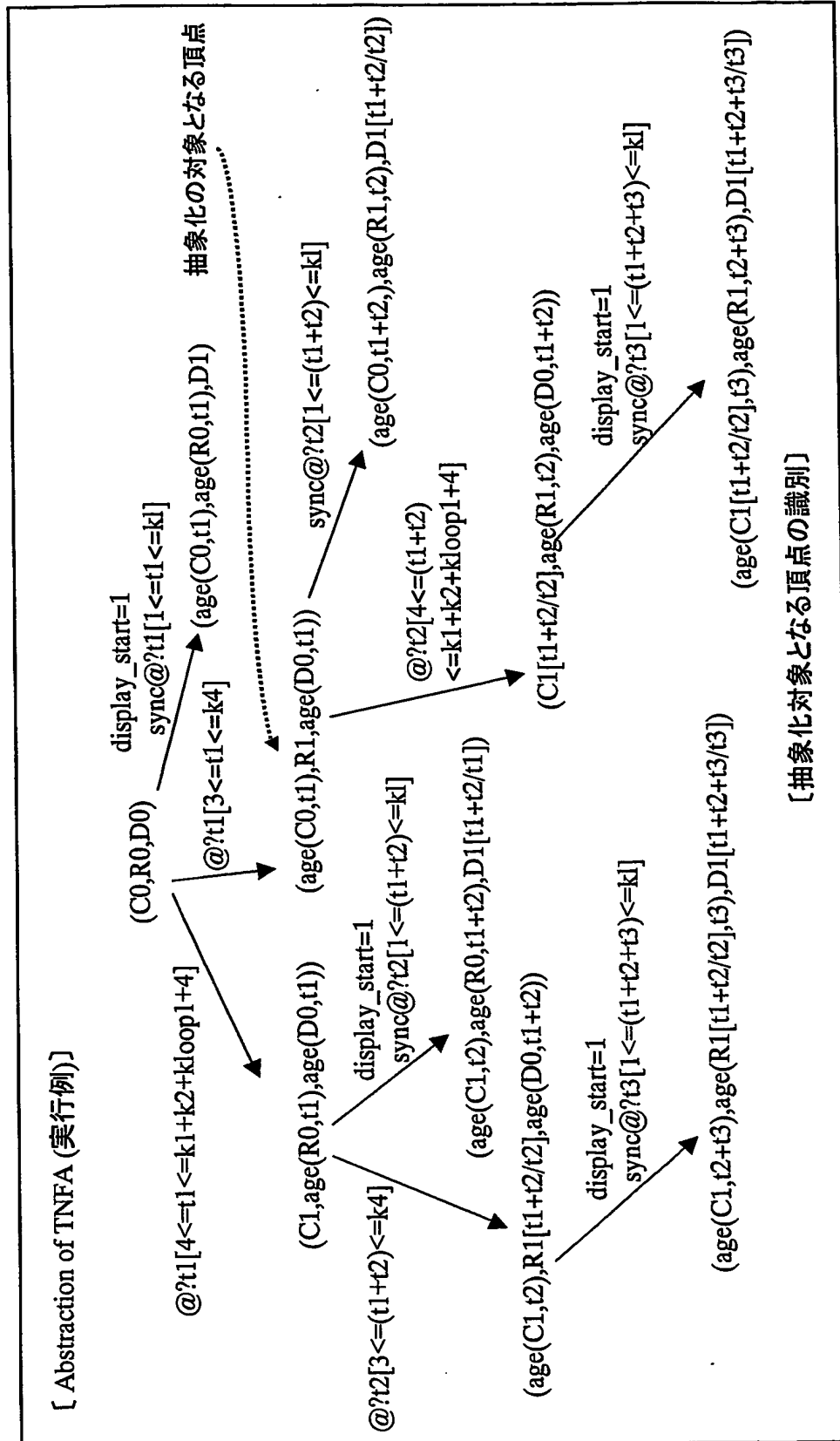


64 / 75

第77図

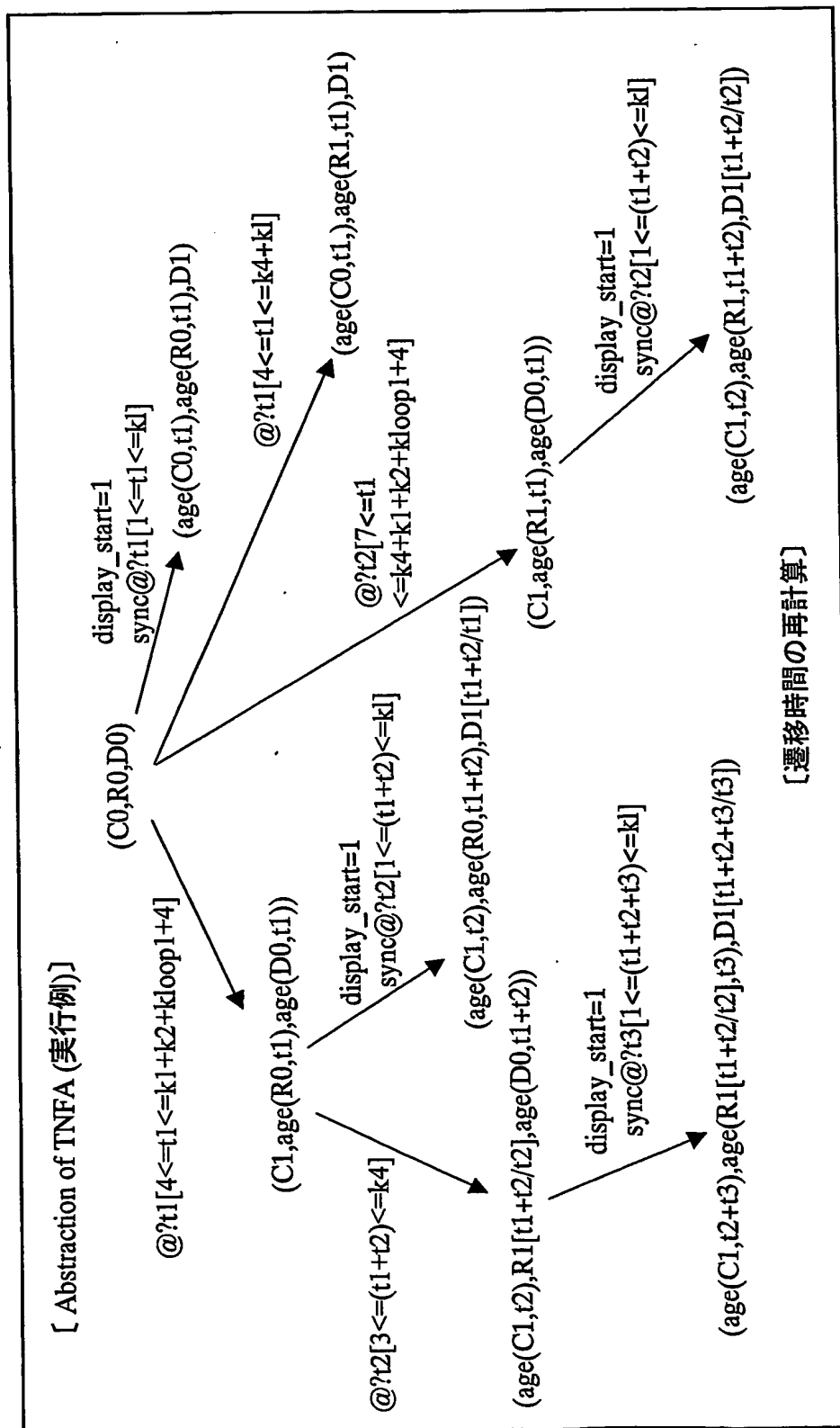


第 7 8 図



66 / 75

第79図



67/75

## 第81図

〔パラメトリック解析結果(実行結果例)〕

Value of objective function: 12

k1	0
k2	0
k3	0
k4	4
k5	0
kd	0
kl	4
kloop1	0
kb1	2
kb2	1
kb3	1
kr1	0
kr2	0

## 第82図

〔パラメトリック解析結果(実行結果例)〕

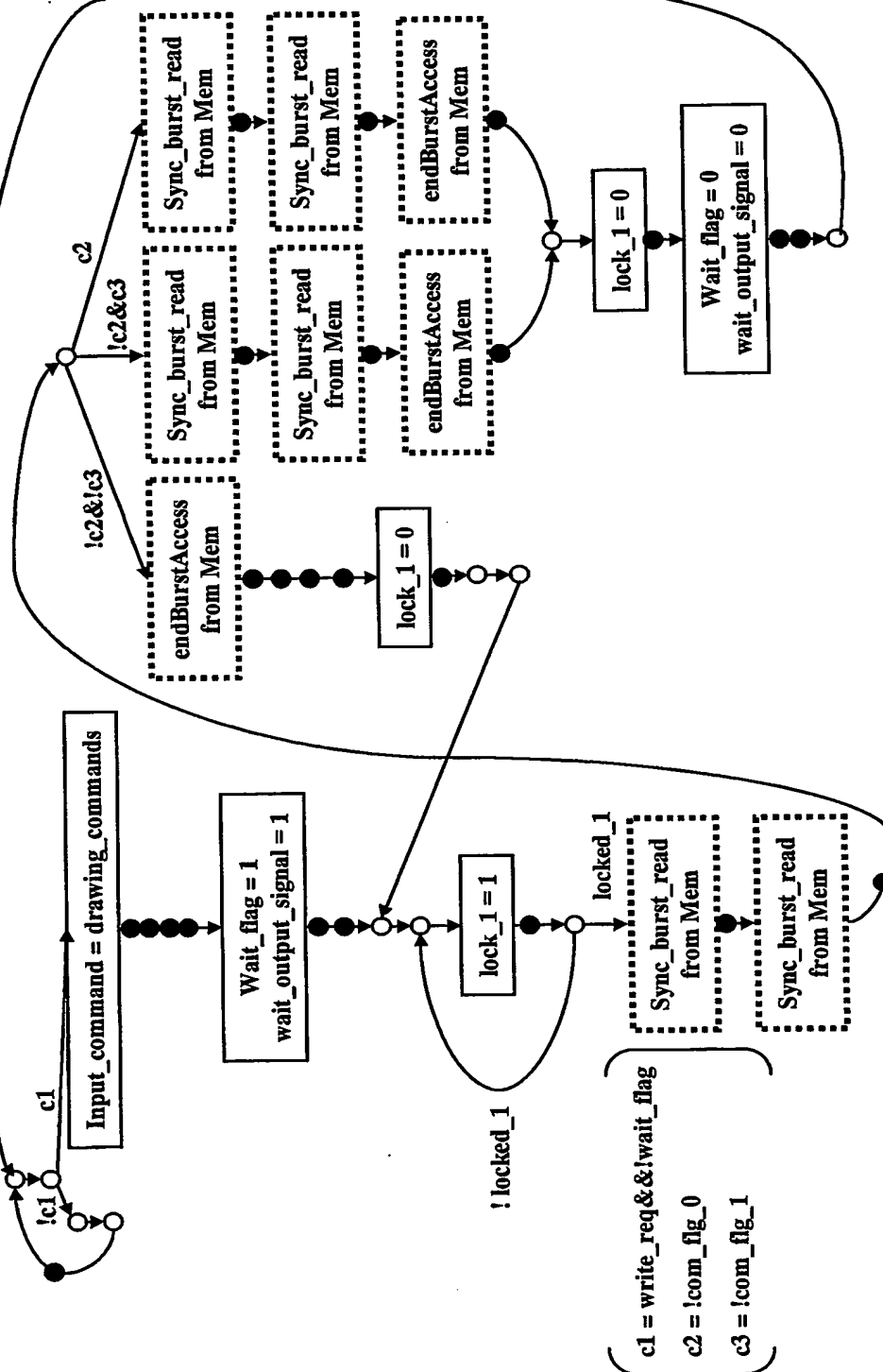
Value of objective function: 19

k1	1
k2	1
k3	1
k4	4
k5	1
kd	1
kl	4
kloop1	1
kb1	2
kb2	1
kb3	1
kr1	0
kr2	1



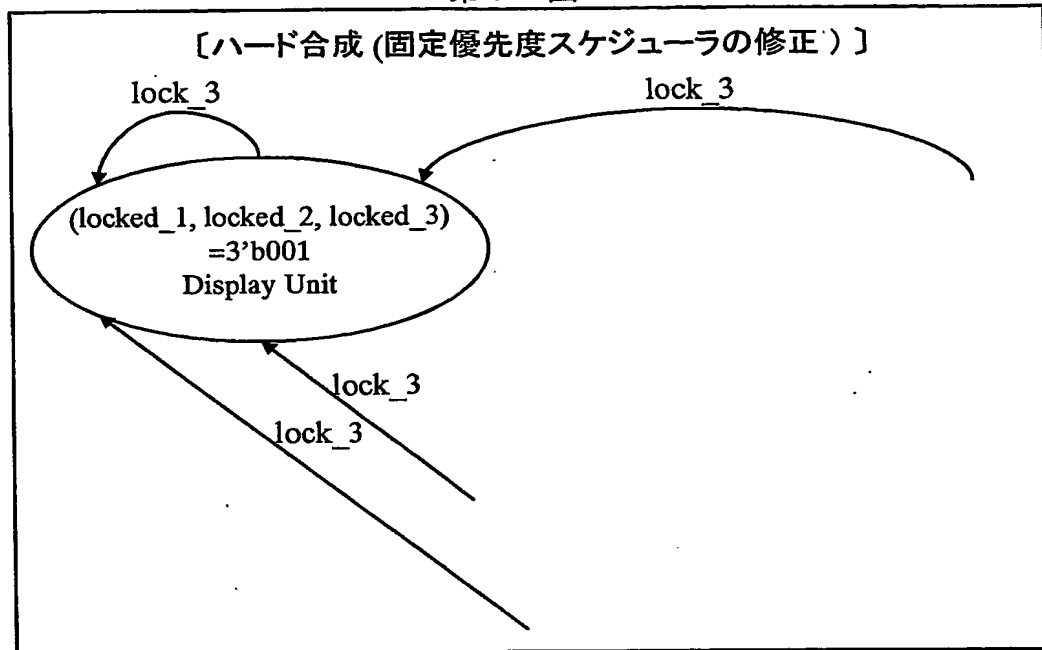
図 38 振

〔ハード合成(BasicBlockへの実行サイクルの割り当て)〕



69 / 75

第84図



第90図

〔ハード合成 (共有レジスタ)〕

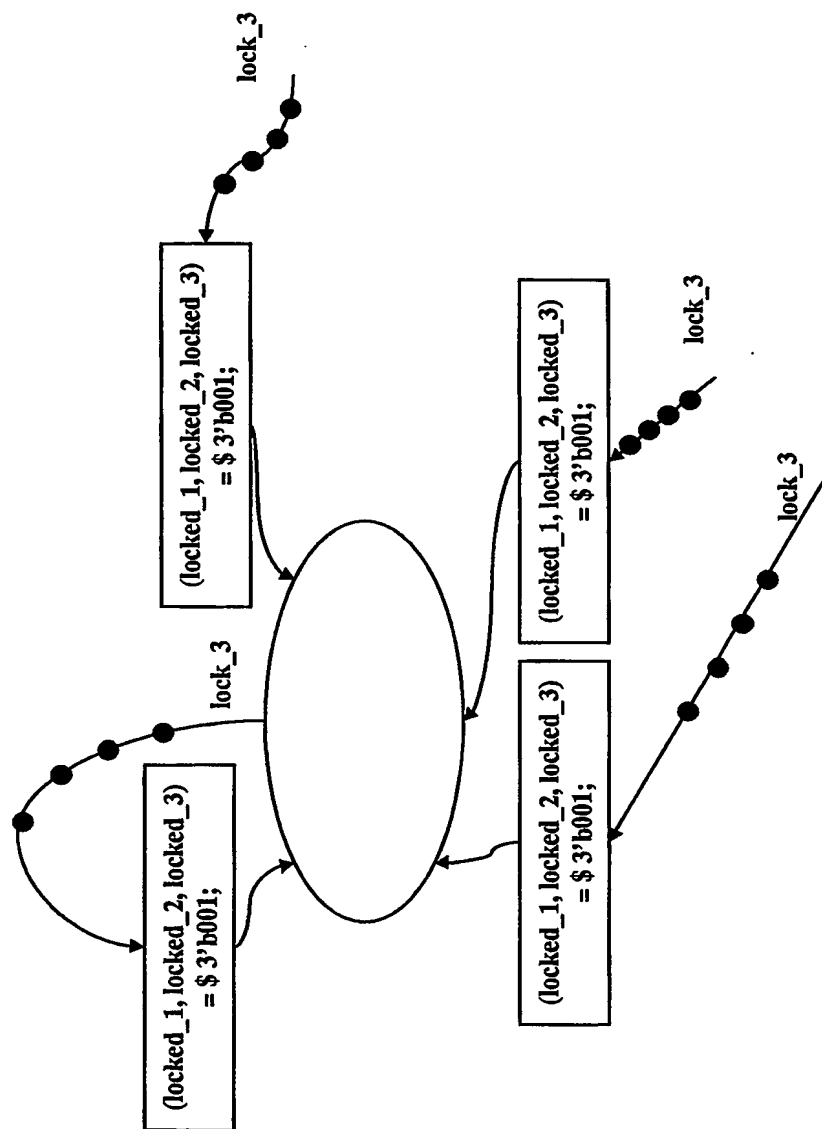
```

void register_in() {
  if (AD_BUS[3:0] == 4'b0000) {
    mem_con_reg.current_value[0] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b0001) {
    mem_con_reg.current_value[1] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b0010) {
    mem_con_reg.current_value[2] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b0011) {
    mem_con_reg.current_value[3] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b0100) {
    mem_con_reg.current_value[4] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b0101) {
    mem_con_reg.current_value[5] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b0110) {
    mem_con_reg.current_value[6] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b0111) {
    mem_con_reg.current_value[7] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b1000) {
    mem_con_reg.current_value[8] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b1001) {
    mem_con_reg.current_value[9] = $D_BUS;
  }
}
  
```

70/75

第85図

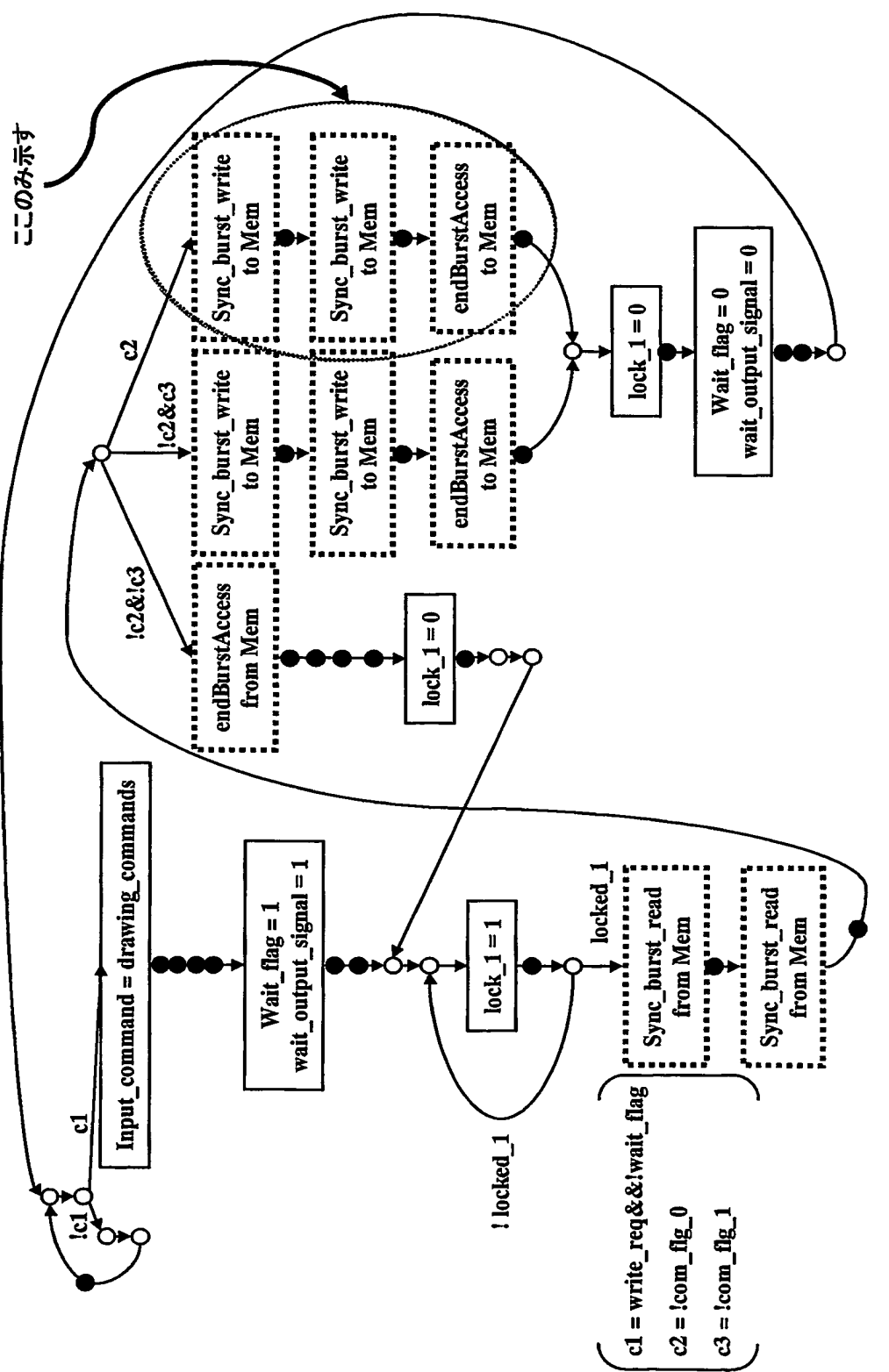
〔ハード合成(固定優先度スケジューラの修正)〕



71 / 75

第 86 図

〔ハード合成(CFGの変形)〕



72/75

第87図

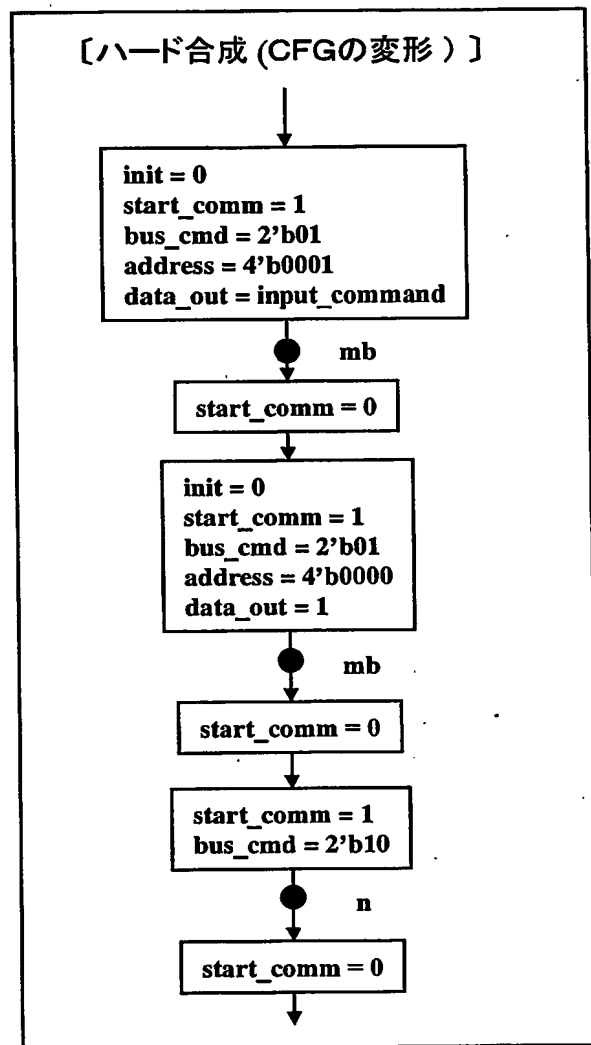
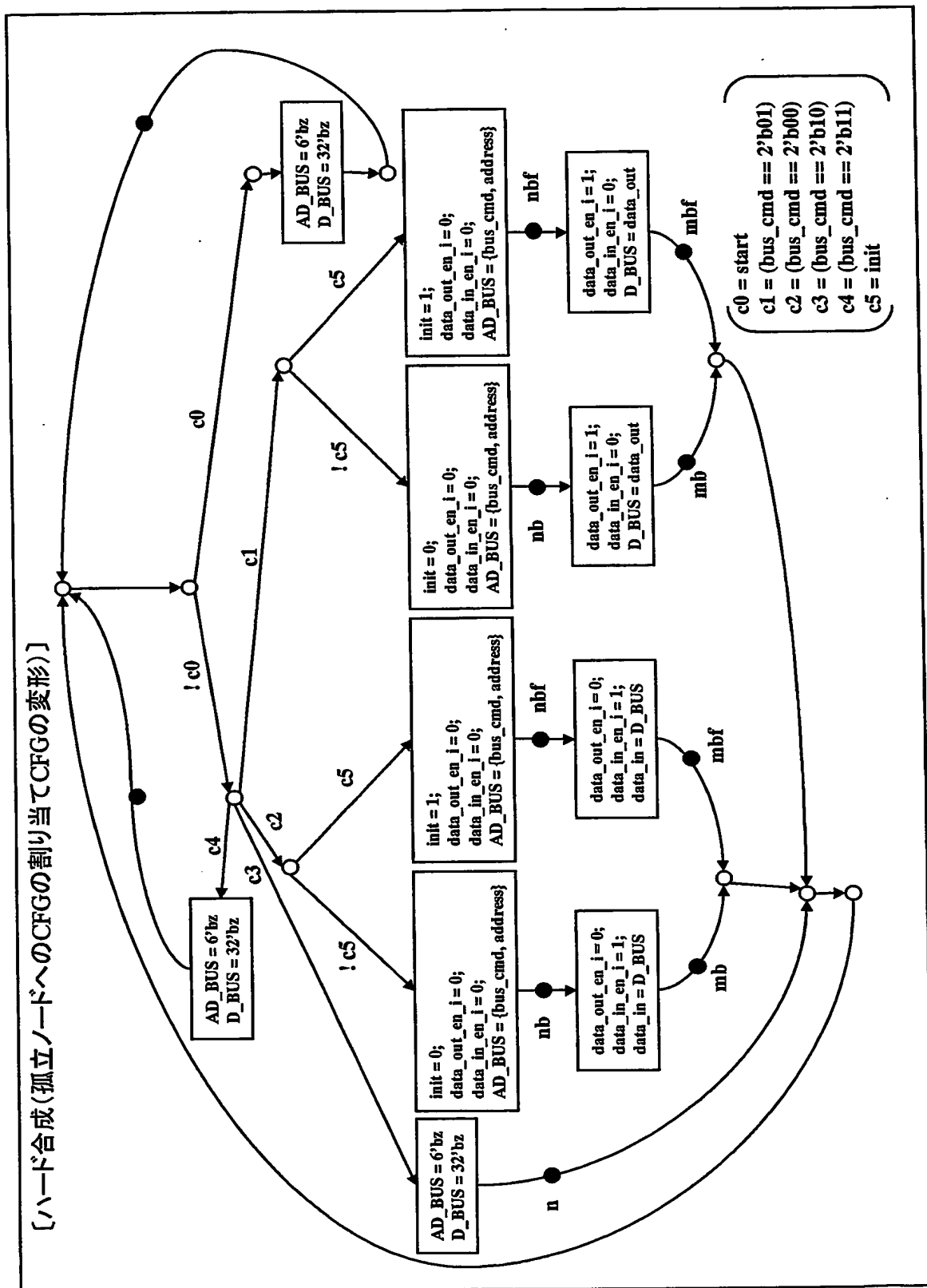


図 8  
8  
8  
振

〔ハード合成(孤立ノードへのCFGの割り当てCFGの変形)〕



74 / 75

## 第89図

【ハード合成(共有レジスタ)】

```

while(1) {
  D_BUS = 32'bz;
  if (locked_1 || locked_2 || locked_3) {
    if (AD_BUS[5:4] == 2'b00) {
      if (init) {
        L1:
          if (data_out_en_1 || data_out_en_2 || data_out_en_3) {
            $
              register_in();
            } else {
              $ goto L1;
            }
          } else {
            L2:
              if (data_out_en_1 || data_out_en_2 || data_out_en_3) {
                register_in();
              } else {
                $ goto L2;
              }
            }
          }
        }
      }
    }
  }
}

```

```

} else if (AD_BUS == 2'b01) {
  if (init) {
    L3:
      if (data_in_en_1 || data_in_en_2 || data_in_en_3) {
        $
          register_out();
        } else {
          $ goto L3;
        }
      } else {
        L4:
          if (data_in_en_1 || data_in_en_2 || data_in_en_3) {
            register_out();
          } else {
            $ goto L4;
          }
        }
      }
    }
  }
}

```

75 / 75

## 第91図

〔ハード合成 (共有レジスタ) 〕

```
void register_out() {  
  if (AD_BUS[3:0] == 4'b0000) {  
    D_BUS = mem_con_reg.current_value[0];  
  } else if (AD_BUS[3:0] == 4'b0001) {  
    D_BUS = mem_con_reg.current_value[1];  
  } else if (AD_BUS[3:0] == 4'b0010) {  
    D_BUS = mem_con_reg.current_value[2];  
  } else if (AD_BUS[3:0] == 4'b0011) {  
    D_BUS = mem_con_reg.current_value[3];  
  } else if (AD_BUS[3:0] == 4'b0100) {  
    D_BUS = mem_con_reg.current_value[4];  
  } else if (AD_BUS[3:0] == 4'b0101) {  
    D_BUS = mem_con_reg.current_value[5];  
  } else if (AD_BUS[3:0] == 4'b0110) {  
    D_BUS = mem_con_reg.current_value[6];  
  } else if (AD_BUS[3:0] == 4'b0111) {  
    D_BUS = mem_con_reg.current_value[7];  
  } else if (AD_BUS[3:0] == 4'b1000) {  
    D_BUS = mem_con_reg.current_value[8];  
  } else if (AD_BUS[3:0] == 4'b1001) {  
    D_BUS = mem_con_reg.current_value[9];  
  }  
}
```



# INTERNATIONAL SEARCH REPORT

International application No.

PCT/JP03/12840

## A. CLASSIFICATION OF SUBJECT MATTER

Int.Cl<sup>7</sup> G06F17/50

According to International Patent Classification (IPC) or to both national classification and IPC

## B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

Int.Cl<sup>7</sup> G06F17/50

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

## C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y.	Masayuki IENAGA et al., "Seigyo Shori Hardware no Koi Gosei no Tame no Kosoku na Menseki/Jikan Saitekika Algorithm", DA Symposium 2000, Information Processing Society of Japan, 17 July, 2000 (17.07.00), Vol.2000, No.8, pages 27 to 32	1-4, 6
Y	NAKATA, A. et al., "Deriving Parameter Conditions for Periodic Timed Automata Satisfying Real-Time Temporal Logic Formulas", Proc. of IFIP TC6/WG6. 1 Int.Conf. on Formal Techniques for Networked and Distributed Systems (FORTE2001), Kluwer Academic Publishers, 2001.08, pages 151 to 166	1-4, 6
Y	GB 2317245 A (SHARP KABUSHIKI KAISHA), 18 March, 1998 (18.03.98), Full text & JP 10-116302 A	1-4, 6

☒ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:	"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A" document defining the general state of the art which is not considered to be of particular relevance	"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E" earlier document but published on or after the international filing date	"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&" document member of the same patent family
"O" document referring to an oral disclosure, use, exhibition or other means	
"P" document published prior to the international filing date but later than the priority date claimed	

Date of the actual completion of the international search  
06 November, 2003 (06.11.03)

Date of mailing of the international search report  
25 November, 2003 (25.11.03)

Name and mailing address of the ISA/  
Japanese Patent Office

Authorized officer

Facsimile No.

Telephone No.

## INTERNATIONAL SEARCH REPORT

International application No.

PCT/JP03/12840

## C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	Kazutoshi WAKABAYASHI et al., "Densoyo LSI o Dosa Gosei de Kaihatsu, Kino Sekkei no Kikan ga 1/10 ni Tanshuku", Nikkei Electronics, Nikkei Business Publications, Inc., 12 February, 1996 (12.02.96), No.655, pages 147 to 169	1-4, 6
A	Hitoshi OKEWATARI et al., "Multi Processor ni yoru Bunsan Shori o Ishiki Shita Senyo Processor Sekkei Shien System SYARDS no Kochiku", Information Processing Society of Japan Kenkyu Hokoku, Information Processing Society of Japan, 20 January, 1995 (20.01.95), Vol.95, No.6(DA-73), pages 105 to 112	5
P,X	Satoru KITAGUCHI et al., "Jitsujikan Seiyaku o Yusuru Tan'itsu Bus System no JAVA ni yoru Model-ka oyobi Parametric Model Checking o Mochiita Sekkei Shuho no Teian", Information Processing Society of Japan Kenkyu Hokoku, Information Processing Society of Japan, 28 November, 2002 (28.11.02), Vol.2002, No.113(SLDM-107), pages 19 to 24	1-6

## A. 発明の属する分野の分類 (国際特許分類 (IPC))

Int. Cl. 7 G06F17/50

## B. 調査を行った分野

調査を行った最小限資料 (国際特許分類 (IPC))

Int. Cl. 7 G06F17/50

最小限資料以外の資料で調査を行った分野に含まれるもの

国際調査で使用した電子データベース (データベースの名称、調査に使用した用語)

## C. 関連すると認められる文献

引用文献の カテゴリー*	引用文献名 及び一部の箇所が関連するときは、その関連する箇所の表示	関連する 請求の範囲の番号
Y	家長真行、外3名 “制御処理ハードウェアの高位合成のための高速な面積/時間最適化アルゴリズム”、DAシンポジウム2000、情報処理学会、2000.07.17、Vol.2000、No.8、p.27-32	1-4, 6
Y	Nakata, A. et al. "Deriving Parameter Conditions for Periodic Timed Automata Satisfying Real-Time Temporal Logic Formulas", Proc. of IFIP TC6/WG6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems (FORTE2001)、Kluwer Academic Publishers、2001.08、p.151-166	1-4, 6

☒ C欄の続きにも文献が列举されている。☐ パテントファミリーに関する別紙を参照。

## \* 引用文献のカテゴリー

「A」特に関連のある文献ではなく、一般的技術水準を示すもの  
「E」国際出願日前の出願または特許であるが、国際出願日以後に公表されたもの  
「L」優先権主張に疑義を提起する文献又は他の文献の発行日若しくは他の特別な理由を確立するために引用する文献 (理由を付す)  
「O」口頭による開示、使用、展示等に言及する文献  
「P」国際出願日前で、かつ優先権の主張の基礎となる出願

の日の後に公表された文献

「T」国際出願日又は優先日後に公表された文献であって出願と矛盾するものではなく、発明の原理又は理論の理解のために引用するもの  
「X」特に関連のある文献であって、当該文献のみで発明の新規性又は進歩性がないと考えられるもの  
「Y」特に関連のある文献であって、当該文献と他の1以上の文献との、当業者にとって自明である組合せによって進歩性がないと考えられるもの  
「&」同一パテントファミリー文献

国際調査を完了した日

06.11.03

国際調査報告の発送日

25.11.03

国際調査機関の名称及びあて先

日本国特許庁 (ISA/JP)

郵便番号100-8915

東京都千代田区霞が関三丁目4番3号

特許庁審査官 (権限のある職員)

早川 学



5H

9652

電話番号 03-3581-1101 内線 3531

C (続き) . 関連すると認められる文献		
引用文献の カテゴリー*	引用文献名 及び一部の箇所が関連するときは、その関連する箇所の表示	関連する 請求の範囲の番号
Y	GB 2317245 A (SHARP K.K.) 1998. 03. 18、全文 & JP 10-116302 A	1-4, 6
Y	若林一敏、外 7 名、“伝送用 L S I を動作合成で開発、機能設計の 期間が 1 / 1 0 に短縮”、日経エレクトロニクス、日経 B P 社、 1996. 02. 12、No. 655、p. 147-169	1-4, 6
A	桶渡仁、外 1 名、“マルチプロセッサによる分散処理を意識した専 用プロセッサ設計支援システム S Y A R D S の構築”、情報処理学 会研究報告、情報処理学会、1995. 01. 20、Vol. 95、No. 6 (DA-73)、 p. 105-112	5
P X	北口智、外 3 名、“実時間制約を有する単一バスシステムの J a v a によるモデル化およびパラメトリックモデルチェック を用いた設計手法の提案”、情報処理学会研究報告、情報処理学 会、2002. 11. 28、Vol. 2002、No. 113 (SLDM-107)、p. 19-24	1-6